



Umm Al Qura University  
Collage of Computer and Information System.  
Computer Science Dept

# Expert System

## CLIPS Basic Student Guide

( C Language Integrated Production System )

Louai m AlArabi

November 2010



CLIPS Basic Student Guide by ALARABI, LOUAI MISHAL H is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License.

Site: <https://sites.google.com/site/orabilouai/my-library/expert-system-clips>

## Expert System – Basic Student guide fro C Language Integrated Production System CLIPS

### What is this document made for ?

In November 2010 I teach CLIPS in Lab of UQU with Porf.Osama Khozium.

This document made for helping students to learn the basic for CLIPS, this documents has some examples program and their execution.

By the end of this document you will learn how to integrate CLIPS with one of the following programming languages C , C++ , C# , Python and Java.

### Is this document enough for Students ?

For whom is interesting of learning CLIPS for non Engineer this document consider as an easy open door to the world of Computation.

Engineering field such Computer Science , Computer Engineering and other this is not enough to be a novel you must use the reference of CLIPS language found in the following website:

<http://clipsrules.sourceforge.net/>.

This document will cover some of the Basic programming in CLIPS not all.

### What tools should I use for building Expert System with CLIPS ?

- CLIPS :

Is an expert system tool developed by the Software Technology Branch (STB) at the NASA/ Lyndon B. Johnson Space Center. It was released in 1986.

Website: <http://clipsrules.sourceforge.net/>

License: Open source – Public Domain

- JESS :

Is a rule engine and scripting environment written entirely in Sun's Java language by Ernest Friedman-Hill at Sandia National Laboratories, it was first written in late 1995, its powerful scripting language give you the access to all of Java APIs.

Website: <http://www.jessrules.com/>

License: Closed Source – Public Domain

### How do you read this document ?

**First** of all I hope you will earn the good benefit from this document.

**Secondly** you need to practice your hand to write code that is the best way to learn any Programming language also you need to visit site to download Example :

<https://sites.google.com/site/orabilouai/my-library/expert-system-clips>

**Finally** to learn more about CLIPS read the official document <http://clipsrules.sourceforge.net/>.

Louai m Alarabi,  
Teacher Assistance, UQU  
Computer Science Dept  
[lmarabi@uqu.edu.sa](mailto:lmarabi@uqu.edu.sa)

# INTRODUCTION

CLIPS, Concern about how to represent (codifying) a Knowledge into Knowledge Base (KB) so can be access late by an expert system.

Before you start reading this document please make sure that you got a good answer for all these question, you can search about them, personally I recommended to discuss this with your Professor.

What is Data ?

What is Information?

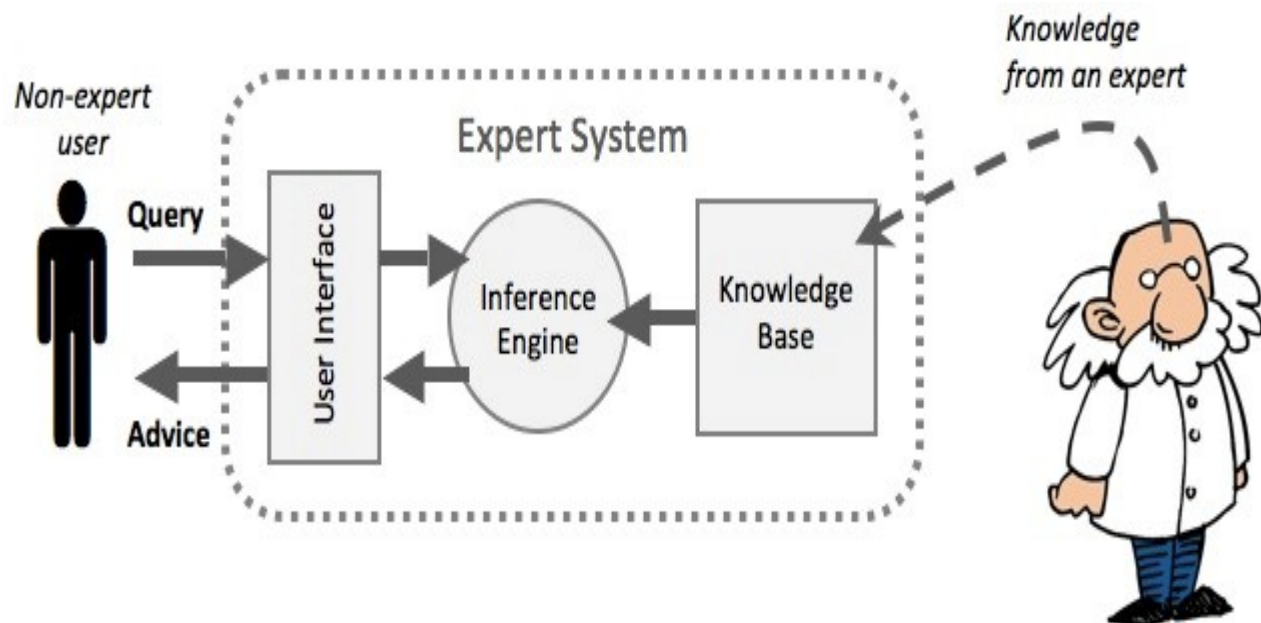
What is Knowledge ?

Why representing Knowledge?

What is different between Knowledge base and Database ?

What makes Expert System special from any another system ?

What is inference Engine and what technique it provide ?



# CLIPS OVERVIEW



Write Hello World in CLIPS

```
CLIPS
File Edit View Terminal Help
CLIPS> (printout t "Hello to CLIPS World" crlf)
Hello to CLIPS World
CLIPS>
```

t – mean Terminal.

Crlf – mean carriage returned line feed.

CLIPS commands are always encased in brackets thus: **(assert (foo))**. Here is a list of some important commands:

(exit)	Shuts down CLIPS
(clear)	Remove all Rule and Facts from memory. Equivalent to shutting down and restarting CLIPS
(reset)	Remove facts from memory (but not Rule) and reset the agenda
(run)	Start executing CLIPS program.
; comments	Semi colon used for comments.



Data Type:

Float	Integer	Symbol	String	External-Address	Fact-Address	Instance-Name	Instance-Address
21.2 32.3e-1	43	say_hi 823sfd %\$^&	“say hi”	<Pointer-X>	<Fact-X>	[say_hi]	<Instance-X>

Q: Is there a Boolean data type in CLIPS, if your answer is yes or even no please give an example ? (discuss this with your teacher or colleagues )

Ans:



Basic I/O command in CLIPS :

## Save File:

If the file saved successfully then the function return TRUE otherwise return FALSE and might some explain log .

Syntax:

(save <file-name>)

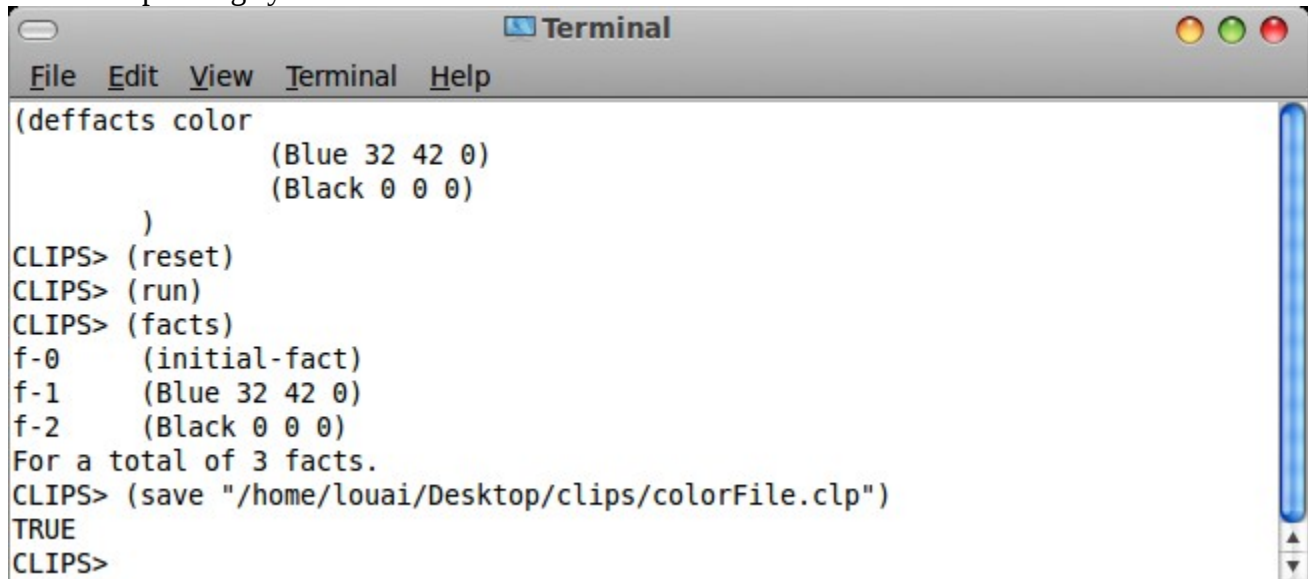
Example:

on windows operating system.



```
CLIPS> (deffacts color
      (Red 22 23 0)
      (Black 0 0 0)
    )
CLIPS> (save "C:\\\\Clips\\\\FileColor.clp")
TRUE
CLIPS> |
```

On Linux operating system



```
(deffacts color
      (Blue 32 42 0)
      (Black 0 0 0)
    )
CLIPS> (reset)
CLIPS> (run)
CLIPS> (facts)
f-0      (initial-fact)
f-1      (Blue 32 42 0)
f-2      (Black 0 0 0)
For a total of 3 facts.
CLIPS> (save "/home/louai/Desktop/clips/colorFile.clp")
TRUE
CLIPS>
```

## Open File:

IF the file open successfully then function return TRUE otherwise return False and might be there some explained log.

Syntax :

(open <file-name> <logical-name> [<mode>])

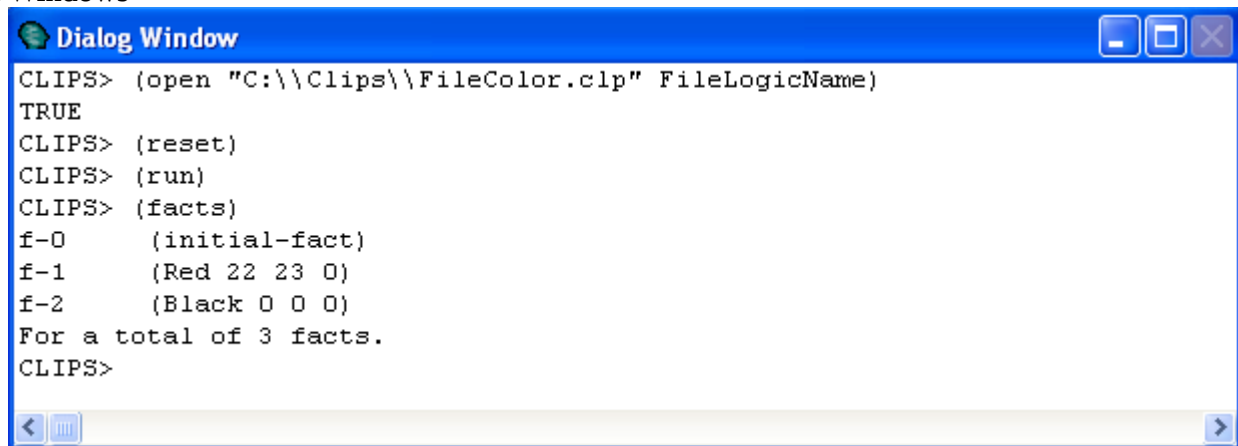
Mode	Mean
"r"	Read access only
"w"	Write access only
"r+"	Read and Write access
"a"	Append access only
"wb"	Binary write access

Example:

(open "studentFactsFile.clp" studentFile "r+")

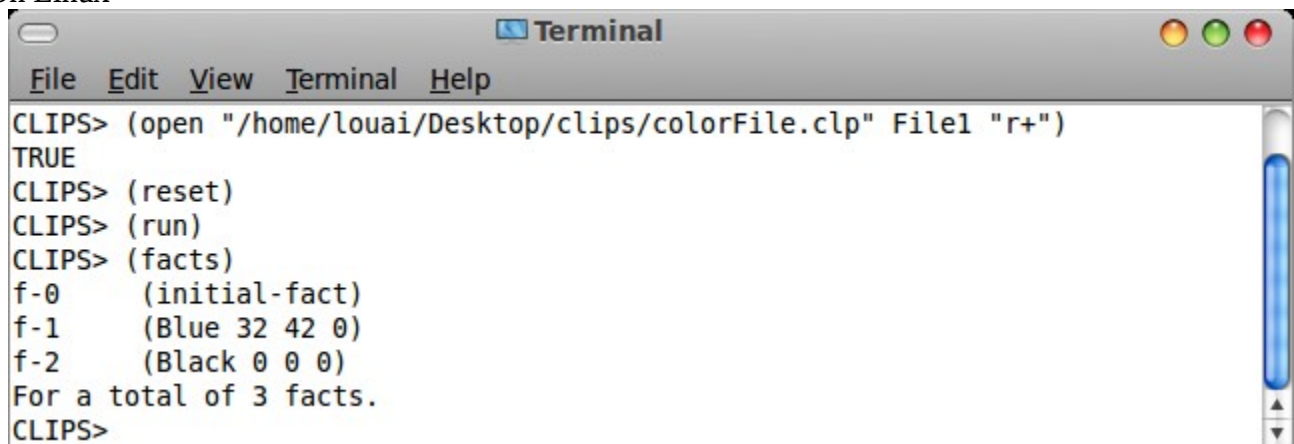
(open "home/clipsProgram/studentFactsFile.clp" Filestd)

On Windows



```
Dialog Window
CLIPS> (open "C:\\Clips\\FileColor.clp" FileLogicName)
TRUE
CLIPS> (reset)
CLIPS> (run)
CLIPS> (facts)
f-0      (initial-fact)
f-1      (Red 22 23 0)
f-2      (Black 0 0 0)
For a total of 3 facts.
CLIPS>
```

On Linux



```
Terminal
File Edit View Terminal Help
CLIPS> (open "/home/louai/Desktop/clips/colorFile.clp" File1 "r+")
TRUE
CLIPS> (reset)
CLIPS> (run)
CLIPS> (facts)
f-0      (initial-fact)
f-1      (Blue 32 42 0)
f-2      (Black 0 0 0)
For a total of 3 facts.
CLIPS>
```

## Close File:

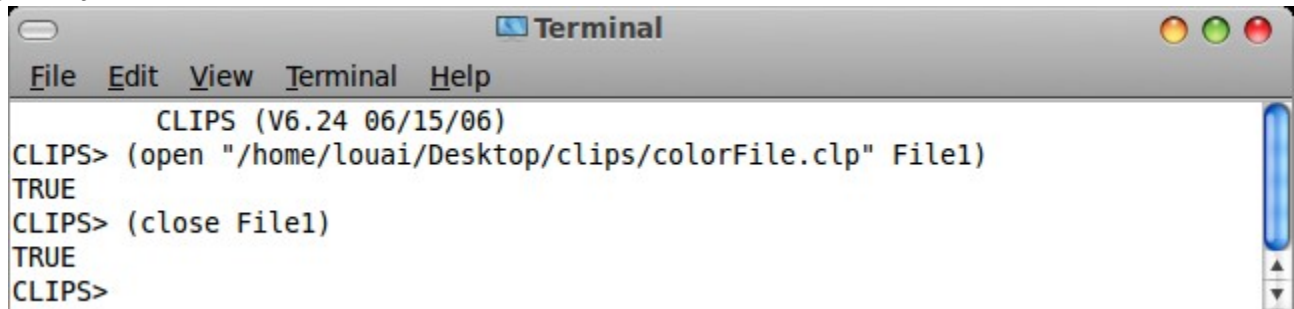
Closing file that previously opened throw command.

Syntax:

`(close <logical-name>)`

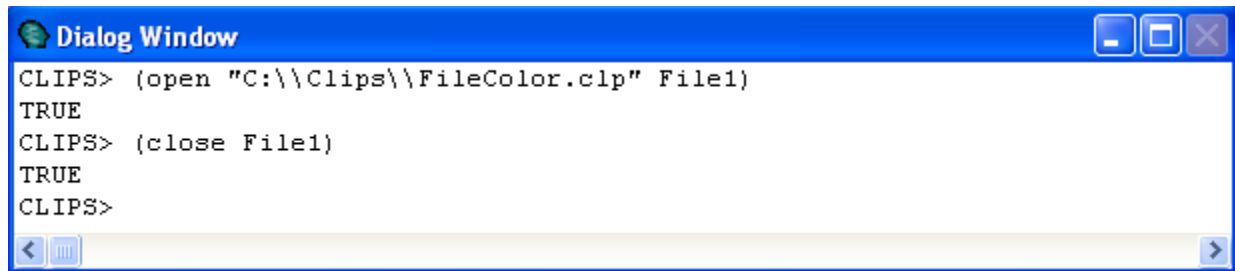
Example:

On Linux

A screenshot of a Linux terminal window titled "Terminal". The window has a menu bar with "File", "Edit", "View", "Terminal", and "Help". The terminal output shows the CLIPS version "CLIPS (V6.24 06/15/06)". The user enters the command "(open \"/home/louai/Desktop/clips/colorFile.clp\" File1)", and the terminal responds with "TRUE". The user then enters "(close File1)", and the terminal again responds with "TRUE". The prompt "CLIPS>" is visible at the bottom.

```
CLIPS (V6.24 06/15/06)
CLIPS> (open "/home/louai/Desktop/clips/colorFile.clp" File1)
TRUE
CLIPS> (close File1)
TRUE
CLIPS>
```

On Windows

A screenshot of a Windows "Dialog Window" titled "Dialog Window". The window has a blue title bar and standard Windows window controls (minimize, maximize, close). The text inside the window shows the same CLIPS commands and output as the Linux terminal: "CLIPS> (open \"C:\\\\Clips\\\\FileColor.clp\" File1)", "TRUE", "CLIPS> (close File1)", "TRUE", and "CLIPS>". There are scroll bars on the right and bottom of the text area.

```
CLIPS> (open "C:\\\\Clips\\\\FileColor.clp" File1)
TRUE
CLIPS> (close File1)
TRUE
CLIPS>
```

## Load File:

Load – is used to load constructs from file into the environment.

Load\* - has the same functionality of load but no information message will appear the function will return TRUE if file loaded successfully otherwise return False.

Bload – is used to load binary constructs file that currently saved using (bsave <file-name>).

Syntax:

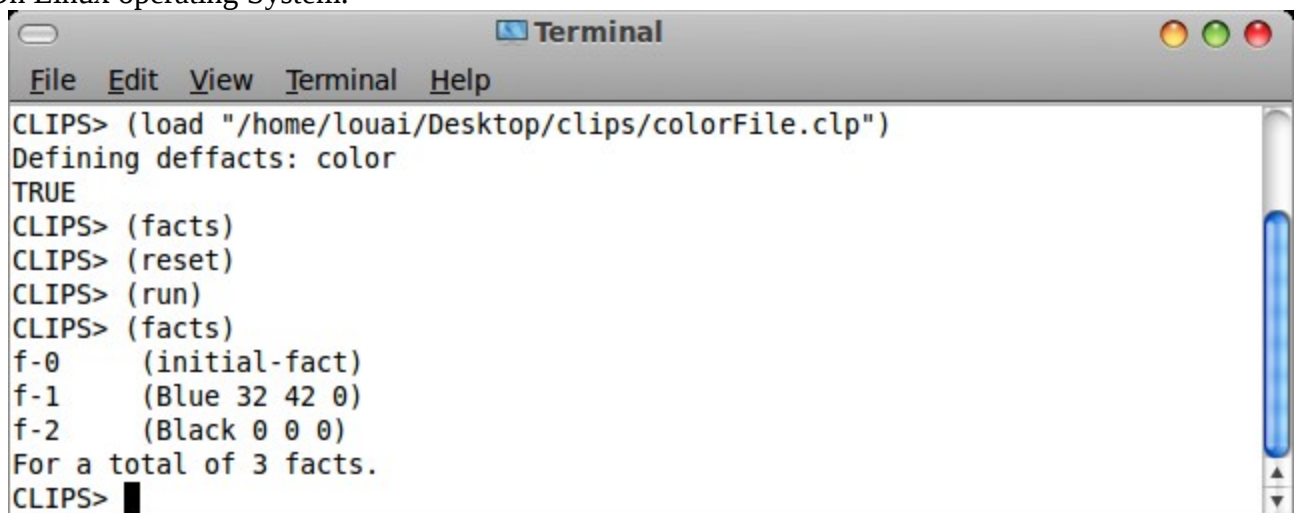
( load <file-name> )

( load\* <file-name> )

( bload <file-name> )

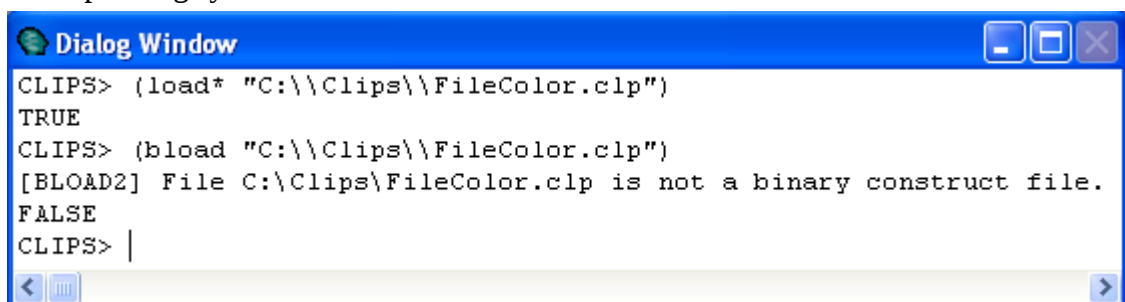
Example:

On Linux operating System.

A screenshot of a Linux terminal window titled "Terminal". The window has a menu bar with "File", "Edit", "View", "Terminal", and "Help". The terminal shows the following interaction:

```
CLIPS> (load "/home/louai/Desktop/clips/colorFile.clp")
Defining deffacts: color
TRUE
CLIPS> (facts)
CLIPS> (reset)
CLIPS> (run)
CLIPS> (facts)
f-0      (initial-fact)
f-1      (Blue 32 42 0)
f-2      (Black 0 0 0)
For a total of 3 facts.
CLIPS> █
```

On Windows operating system.

A screenshot of a Windows "Dialog Window" titled "Dialog Window". The window has standard Windows window controls (minimize, maximize, close). The text inside shows the following interaction:

```
CLIPS> (load* "C:\\Clips\\FileColor.clp")
TRUE
CLIPS> (bload "C:\\Clips\\FileColor.clp")
[BLOAD2] File C:\\Clips\\FileColor.clp is not a binary construct file.
FALSE
CLIPS> |
```

# KNOWLEDGE REPRESENTATION



## FACTS:

deffacts constructs, assert a list of facts.

Syntax:

```
(deffacts <deffacts-name> [<comment>]  
  <RHS-pattern>*)
```

Example

```
(deffacts home "This is facts about home"  
  (door is open)  
)
```

The table below with the name of std has the following facts:

Name	Age	Gender
Ali	24	M
Omar	19	M
Nouf	32	F
Saed	21	M
Sara	22	F

```
CLIPS (V6.24 06/15/06)  
CLIPS> (deffacts std "facts about students"  
  (std Ali 24 M)  
  (std Omar 19 M)  
  (std Nouf 32 F)  
  (std Saed 21 M)  
  (std Sara 22 F)  
)
```

after writing the facts we would like to **Execute** the program to execute we must use the following command.

```
(reset)
```

```
(run)
```

and for removing facts from working memory we use the following command

```
(retract <fact-number>*) ; * could take more than one parameter.
```

```
louai@louai-laptop: ~  
File Edit View Terminal Tabs Help  
Terminal louai@louai-laptop: ~  
louai@louai-laptop:~$ clips  
CLIPS (V6.24 06/15/06)  
CLIPS> (deffacts std "facts about students"  
          (std Ali 24 M)  
          (std Omar 19 M)  
          (std Nouf 32 F)  
          (std Saed 21 M)  
          (std Sara 22 F)  
        )  
CLIPS> (reset)  
CLIPS> (run)  
CLIPS> (facts)  
f-0      (initial-fact)  
f-1      (std Ali 24 M)  
f-2      (std Omar 19 M)  
f-3      (std Nouf 32 F)  
f-4      (std Saed 21 M)  
f-5      (std Sara 22 F)  
For a total of 6 facts.  
CLIPS>
```



## Rules:

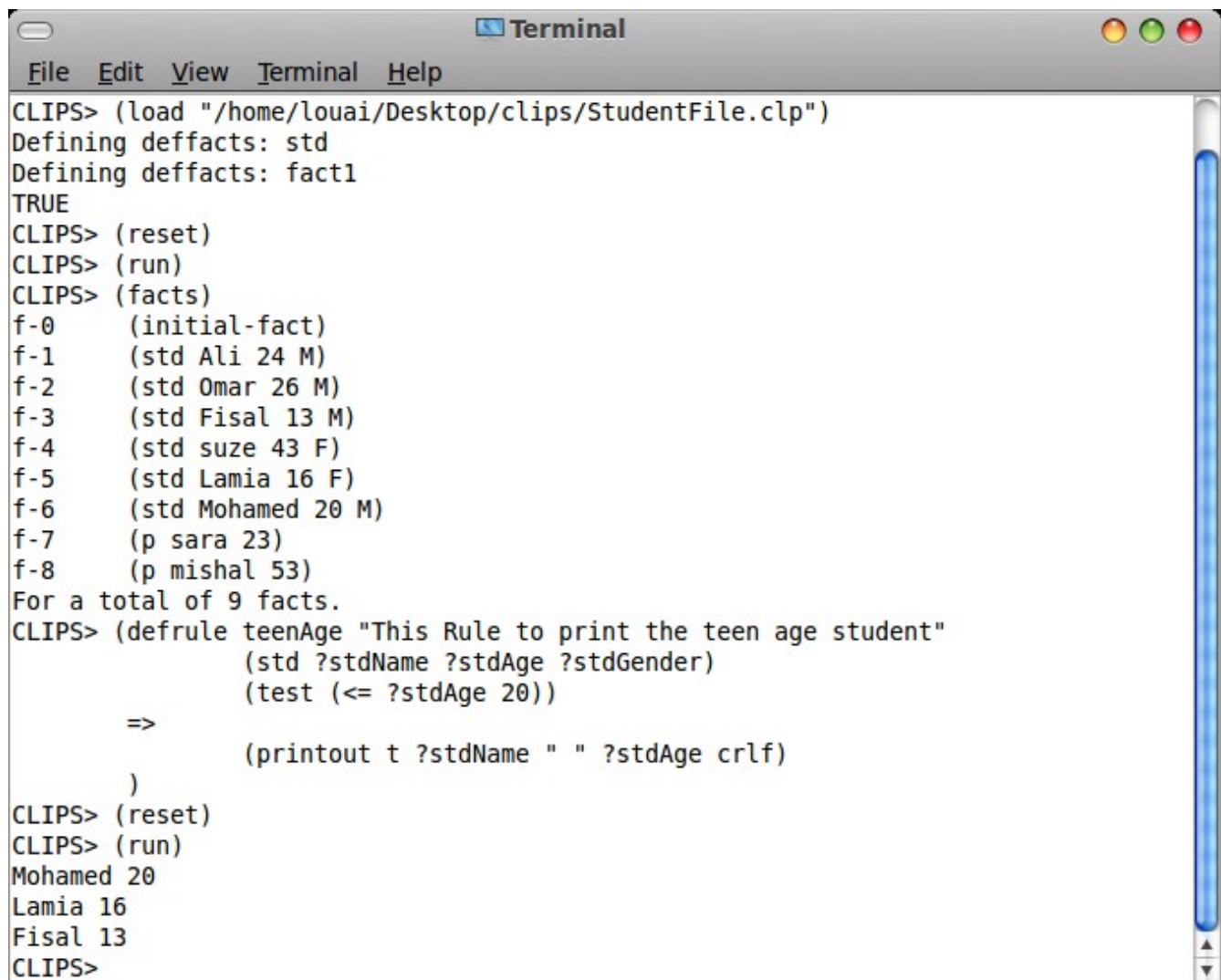
One of the primary method to represents knowledge in CLIPS is a rule, A rule is a collection of condition and action to be taken if the condition met.

### Syntax:

```
(defrule <rule-name> [<comment>]
  [<declaration>]           ; Rule Properties
  <conditional-element>*    ; Left-Hand Side (LHS)
  =>
  <action>*                 ; Right-Hand Side (RHS)
)
```

Rule properties could have any name

Example:



```
CLIPS> (load "/home/louai/Desktop/clips/StudentFile.clp")
Defining deffacts: std
Defining deffacts: fact1
TRUE
CLIPS> (reset)
CLIPS> (run)
CLIPS> (facts)
f-0      (initial-fact)
f-1      (std Ali 24 M)
f-2      (std Omar 26 M)
f-3      (std Fisal 13 M)
f-4      (std suze 43 F)
f-5      (std Lamia 16 F)
f-6      (std Mohamed 20 M)
f-7      (p sara 23)
f-8      (p mishal 53)
For a total of 9 facts.
CLIPS> (defrule teenAge "This Rule to print the teen age student"
      (std ?stdName ?stdAge ?stdGender)
      (test (<= ?stdAge 20))
      =>
      (printout t ?stdName " " ?stdAge crlf)
)
CLIPS> (reset)
CLIPS> (run)
Mohamed 20
Lamia 16
Fisal 13
CLIPS>
```

Exercise:

1- Write a CLIPS program using the previous list of facts and print female student information with count.

Output would look like this:

#1 suze 43 F

#2 lamia 16 F

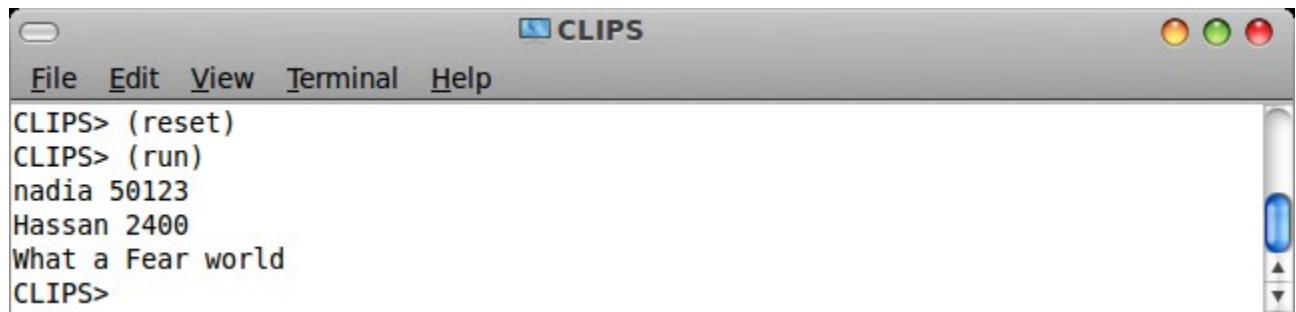
Example 2:

Represents the below facts in Knowledge base and find out how miss the rage of the salary according to employee major.

Major	Minimum salary	Maximum Salary
Computer Science	7000	20000
Biology	4000	9000
Physics	9000	23420

Name	Major	Salary
Abdullah	Computer Science	7300
Hassan	Biology	2400
Lana	Physics	50123

Who has more or less than regular salary the output should be look like this

A screenshot of a CLIPS terminal window. The window has a title bar with the text "CLIPS" and standard window control buttons (minimize, maximize, close). Below the title bar is a menu bar with the options "File", "Edit", "View", "Terminal", and "Help". The main area of the window displays the following text:

```
CLIPS> (reset)
CLIPS> (run)
nadia 50123
Hassan 2400
What a Fear world
CLIPS>
```

```
CLIPS
File Edit View Terminal Help
CLIPS> (deffacts Major
      (m computerScience 7000 20000)
      (m Biology 4000 9000)
      (m Physics 9000 23420)
    )
CLIPS> (deffacts Employee
      (e abdullah computerScience 7300)
      (e Hassan Biology 2400)
      (e nadia Physics 50123)
    )
CLIPS> (defrule Rule1
      (m ?mMajor ?mMini ?mMaxi)
      (e ?eName ?eMajor ?eSalary)
      (test (eq ?mMajor ?eMajor))
      =>
      (if (or (< ?eSalary ?mMini) (> ?eSalary ?mMaxi))
          then
            (printout t ?eName " " ?eSalary crlf)
          else
            (printout t "What a Fear world" crlf)
      )
    )
)
```



Assert

Assert used for adding new facts to the fact list if the fact exist then FALSE will be returned.

Syntax :

(assert <RH-Pattern>)

Example:

The following table has the relation between each father and his son lets call this table M, and the table next table F has relation between Mother and her son. If we supposed that sons in both table are same could you figure out whom married whom. ?

- 1- Represents the facts in table.
- 2- show the facts using (facts) command.
- 3- add Assert to add facts.

Father	Son	Marriage ID
Ali	Ahmed	123
Mohamed	Slma	432
Khaled	Louai	4245
Mosaed	Ali	88808

Mother	Son	Number of Suns
Mona	Nadia	2
Hanan	Ali	4
Eman	Slma	3
Noha	Louai	1

```
CLIPS> (deffacts marriage
      (m Ali Ahmed 123)
      (m Mohamed Slma 432)
      (m Khaled Louai 4245)
      (m Mosaed Ali 88808)
      (f Mona Nadia 2)
      (f Hanan Ali 4)
      (f Eman Slma 3)
      (f Noha Louai 1)
    )
CLIPS> (defrule married "Rule to determin who married who"
      (m ?father ?mSon ?mID)
      (f ?mother ?fSon ?fCount)
      (test (eq ?mSon ?fSon))
    =>
      (assert (married ?father ?mother ?fSon))
    )
CLIPS> (reset)
CLIPS> (run)
CLIPS> (facts)
f-0      (initial-fact)
f-1      (m Ali Ahmed 123)
f-2      (m Mohamed Slma 432)
f-3      (m Khaled Louai 4245)
f-4      (m Mosaed Ali 88808)
f-5      (f Mona Nadia 2)
f-6      (f Hanan Ali 4)
f-7      (f Eman Slma 3)
f-8      (f Noha Louai 1)
f-9      (married Khaled Noha Louai)
f-10     (married Mohamed Eman Slma)
f-11     (married Mosaed Hanan Ali)
For a total of 12 facts.
CLIPS> █
```



User define Function:

deffunction constructs , define a user function like any other language, There is two way to define function in CLIPS:

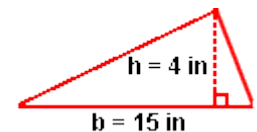
deffunction	Writing in CLIPS and executed by CLIPS
User-defined external function	Writing in other language such C , Java and integrated with CLIPS

Syntax:

```
(deffunction <function-name> [<comment>]
  (<regular-parameter> [<wildcard-parameter>] ) ; parameter
  <action>* ; one or steps of actions
)
```

Example:

- 1) Find the area of triangle with the base of 15 inches and the hight of 4 inches ?  
Hint: Area formula =  $\frac{1}{2}$  (base \* hight)

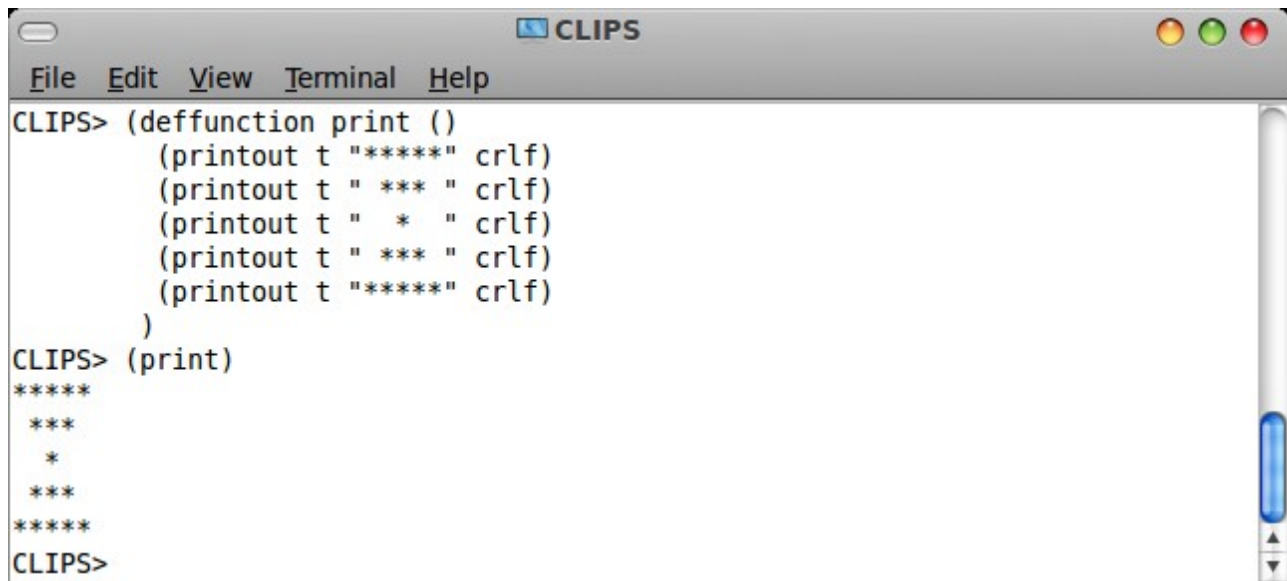


```
CLIPS
File Edit View Terminal Help
CLIPS> (deffunction area-triangle (?base ?height)
  (printout t "Area = " (/ (* ?base ?height) 2) " in*in" crlf)
);end function
CLIPS> (area-triangle 15 4)
Area = 30.0 in*in
CLIPS>
```

- 2) Declare a function that calculate the factorial of number?

```
CLIPS
File Edit View Terminal Help
CLIPS> (deffunction factorial (?a)
  (if (or (not (integerp ?a)) (< ?a 0))
    then
      (printout t "Factorial Error!" crlf)
    else
      (if (= ?a 0)
        then
          1
        else
          (* ?a (factorial (- ?a 1) ) )
      );endif
    );endif
);endFucntion
CLIPS> (factorial 6)
720
```

Notice that the function parameter could be void also an action could be more than one action, see the example below.



```
CLIPS> (deffunction print ()
  (printout t "*****" crlf)
  (printout t " *** " crlf)
  (printout t "  *  " crlf)
  (printout t " *** " crlf)
  (printout t "*****" crlf)
)
CLIPS> (print)
*****
 ***
  *
 ***
*****
CLIPS>
```



Logical Conditional:

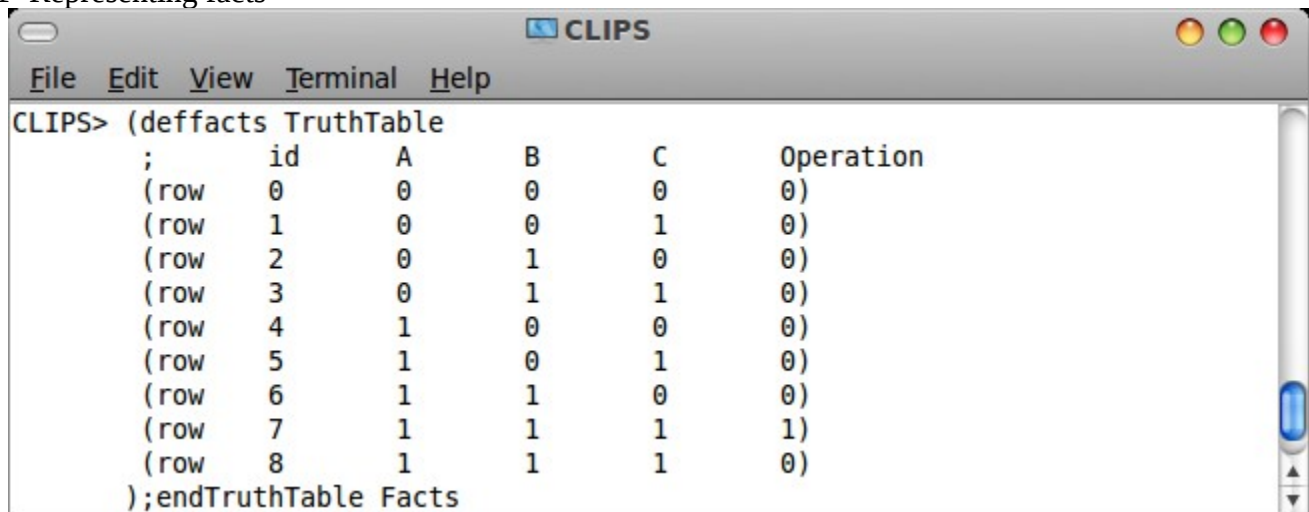
We can use logical conditional in CLIPS with rule , IF ... .

Logical	BNF	Example
Not	(not <conditional-elements>)	(not (= 34 43) )
OR	(or <conditional-elements>)	(or (> ?grade 90) (< ?grade 100))
And	(and <conditional-elements>)	(and (eq ?x ?y) (= ?z (+ ?f 1)))
Is integer	(integerp <expression>)	(integerp 3.2)
Is symoble	(symbolp <expression>)	(symbolp x)
Is string	(stringp <expression>)	(stringp "louai")
Is odd	(oddp <integer-expression>)	(oddp 23)
Is even	(evenp <integer-expression>)	(evenp 25)
Is equal	(eq <expression> <expression>)	(eq "hello" "HELLO")
Is number	(numberp <expression> )	(numberp 34.5) (numberp f23)
IS symbol or string	(lexemep <expression>)	(lexemep x) (lexemep "x") (lexemep ?*globalVar*)

Example: Suppose that we have a below facts in the Truth Table, Write a program that assert a facts from existing ones facts about which facts operation represent Logical AND GATE.

ID	A	B	C	Operation
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	0
4	1	0	0	0
5	1	0	1	0
6	1	1	0	0
7	1	1	1	1

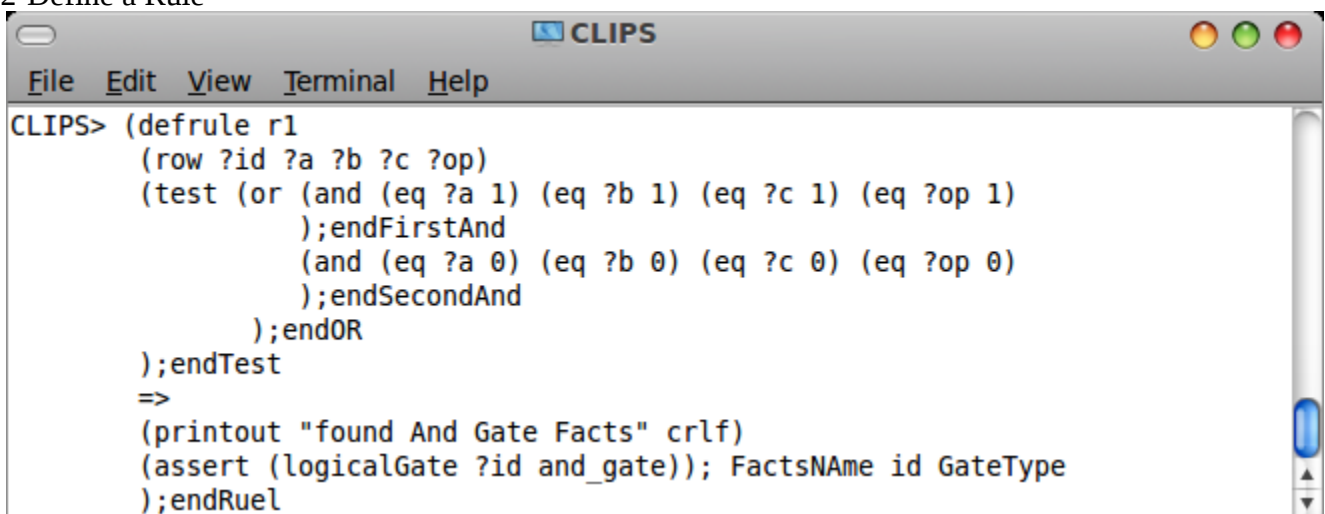
### 1- Representing facts



```

CLIPS> (deffacts TruthTable
;      id      A      B      C      Operation
(row 0      0      0      0      0)
(row 1      0      0      1      0)
(row 2      0      1      0      0)
(row 3      0      1      1      0)
(row 4      1      0      0      0)
(row 5      1      0      1      0)
(row 6      1      1      0      0)
(row 7      1      1      1      1)
(row 8      1      1      1      0)
);endTruthTable Facts
  
```

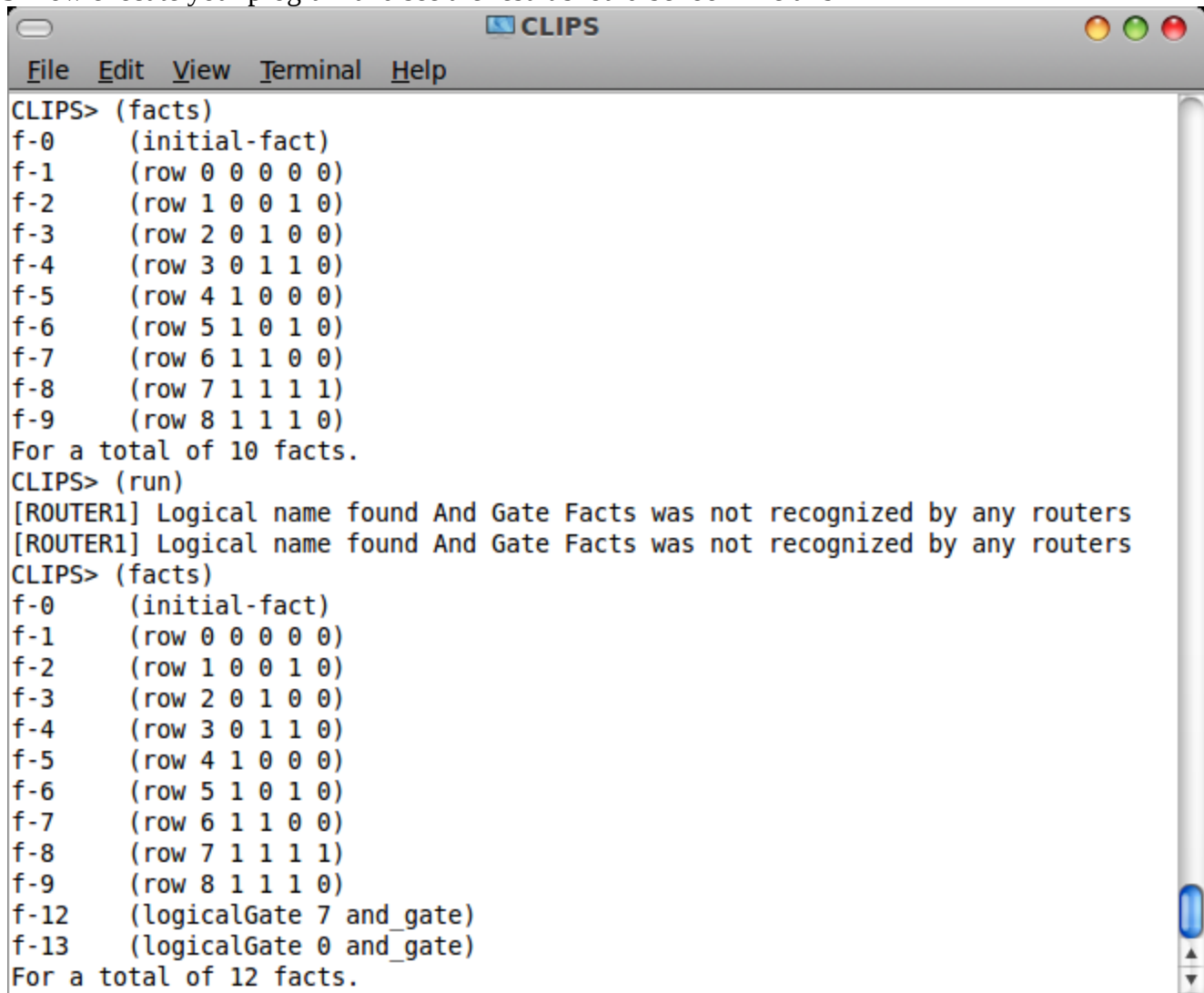
### 2-Define a Rule



```

CLIPS> (defrule r1
  (row ?id ?a ?b ?c ?op)
  (test (or (and (eq ?a 1) (eq ?b 1) (eq ?c 1) (eq ?op 1)
                );endFirstAnd
            (and (eq ?a 0) (eq ?b 0) (eq ?c 0) (eq ?op 0)
                );endSecondAnd
        );endOR
  );endTest
=>
(printout "found And Gate Facts" crlf)
(assert (logicalGate ?id and_gate)); FactsName id GateType
);endRuel
  
```

3- now execute your program and see the result should be look like this



```
CLIPS> (facts)
f-0      (initial-fact)
f-1      (row 0 0 0 0 0)
f-2      (row 1 0 0 1 0)
f-3      (row 2 0 1 0 0)
f-4      (row 3 0 1 1 0)
f-5      (row 4 1 0 0 0)
f-6      (row 5 1 0 1 0)
f-7      (row 6 1 1 0 0)
f-8      (row 7 1 1 1 1)
f-9      (row 8 1 1 1 0)
For a total of 10 facts.
CLIPS> (run)
[ROUTER1] Logical name found And Gate Facts was not recognized by any routers
[ROUTER1] Logical name found And Gate Facts was not recognized by any routers
CLIPS> (facts)
f-0      (initial-fact)
f-1      (row 0 0 0 0 0)
f-2      (row 1 0 0 1 0)
f-3      (row 2 0 1 0 0)
f-4      (row 3 0 1 1 0)
f-5      (row 4 1 0 0 0)
f-6      (row 5 1 0 1 0)
f-7      (row 6 1 1 0 0)
f-8      (row 7 1 1 1 1)
f-9      (row 8 1 1 1 0)
f-12     (logicalGate 7 and_gate)
f-13     (logicalGate 0 and_gate)
For a total of 12 facts.
```

Exercise :

- 1) write a program that assert logical AND, OR , XOR, NOR and NAND gate for the following facts table.

ID	X	Y	Z	Operation
0	0	0	0	1
1	0	0	1	0
2	0	1	0	1
3	0	1	1	0
4	1	0	0	1
5	1	0	1	1
6	1	1	0	0
7	1	1	1	1

Hint: for helping see the Truth table:

A	B	NOT $\neg b$	AND $a \wedge b$	OR $a \vee b$	XOR $a \perp b$	NOR $a \nabla b$	NAND $a \bar{\wedge} b$
FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	TRUE	TRUE
FALSE	TRUE	FALSE	FALSE	TRUE	TRUE	FALSE	TRUE
TRUE	FALSE		FALSE	TRUE	TRUE	FALSE	TRUE
TRUE	TRUE		TRUE	TRUE	FALSE	FALSE	FALSE



Basic Mathematic build in function:

Operation	Example	Result
Addition	(+ 2 3 1)	6
Subtraction	(- 56 3)	53
Multiplication	(* 3 4)	12
Division	(/ 12 2)	6
Division	(div 12 2 3)	2
Maximum	(max 3 3.9 4)	4
Minimum	(min 3 2 1)	1
Absolute value	(abs -2.3)	2.3
Convert to integer	(integer 4.2)	4
Convert to float	(float 9)	9
Square root	(sqrt 36)	6
Power	(** 6 2)	36



Global Variable:

defglobal constructs used for defining a global variable.

Syntax:

(defglobal [<defmodule-name>] <global-assignment>)

<global-assignment> :: = <global-variable> = <expression>

<global-variable> :: = ?\*<symbol>\*

Example:

```

CLIPS
File Edit View Terminal Help
CLIPS> (defglobal ?*32x* = 23)
CLIPS> (+ ?*32x* 1)
24
CLIPS> (numberp ?*32x*)
TRUE
CLIPS> (clear)
CLIPS> (printout t ?*32* crlf)
[GLOBALDEF1] Global variable ?*32* is unbound.
CLIPS>

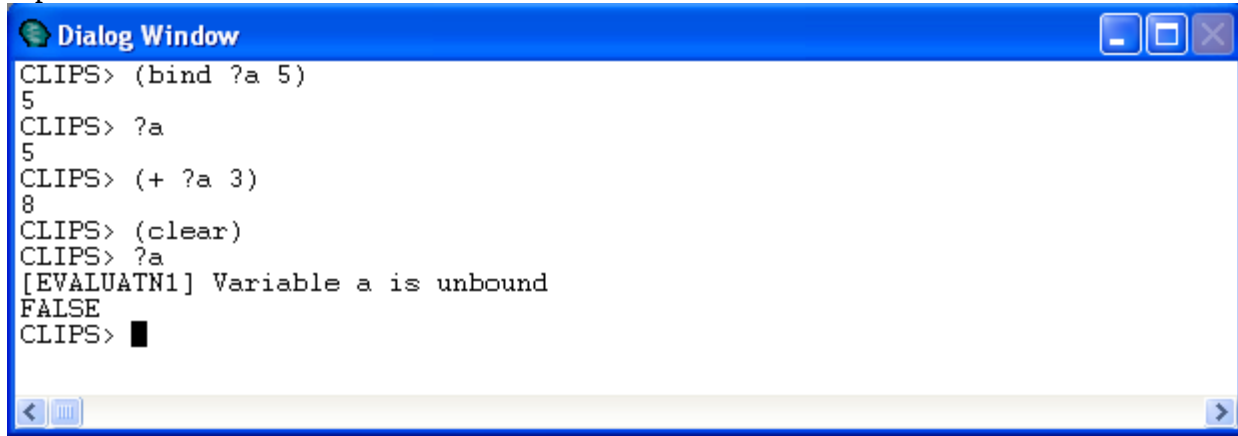
```

▶ Binding Variable:

Syntax:

(bind <Variable-name> <expression>)

Example:



```
CLIPS> (bind ?a 5)
5
CLIPS> ?a
5
CLIPS> (+ ?a 3)
8
CLIPS> (clear)
CLIPS> ?a
[EVALUATN1] Variable a is unbound
FALSE
CLIPS> █
```

▶ Reading from Terminal & File:

Syntax:

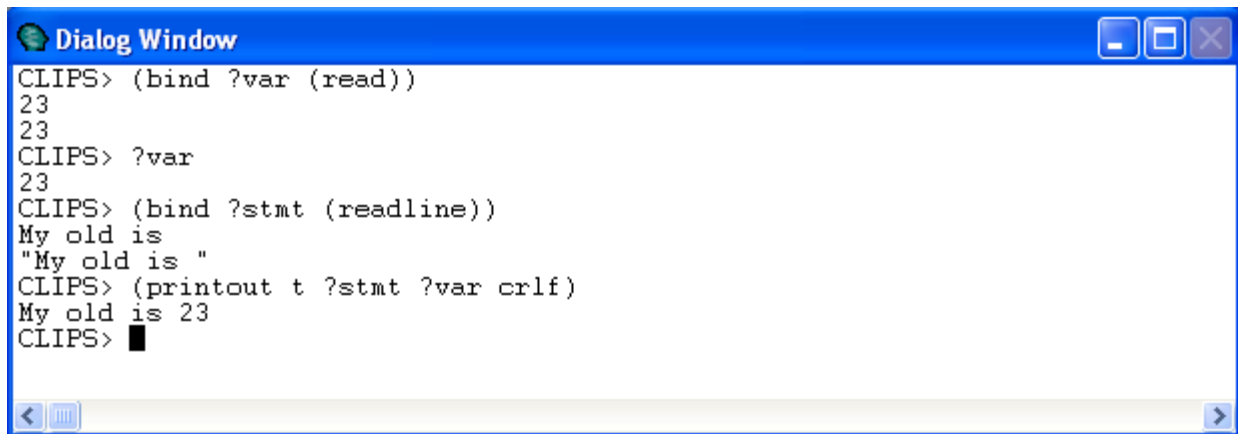
(read [logical-name] )

(readline [logical-name] )

[logical-name] is optional parameter if void then it read from terminal otherwise it read from logical-name.

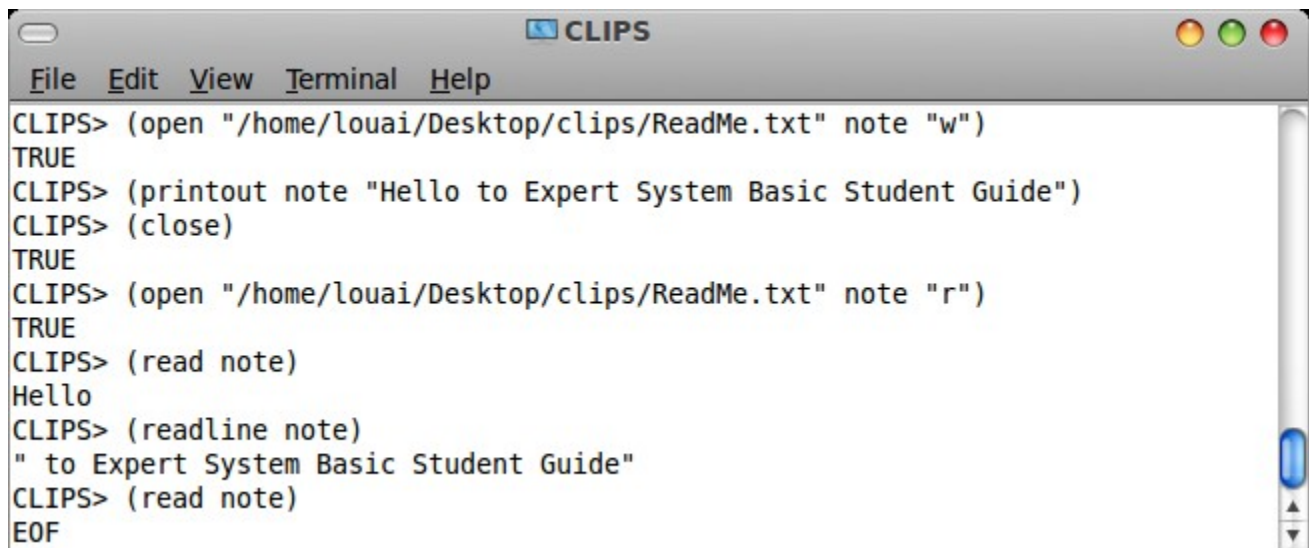
Example:

Reading form Terminal



```
CLIPS> (bind ?var (read))
23
23
CLIPS> ?var
23
CLIPS> (bind ?stmt (readline))
My old is
"My old is "
CLIPS> (printout t ?stmt ?var crlf)
My old is 23
CLIPS> █
```

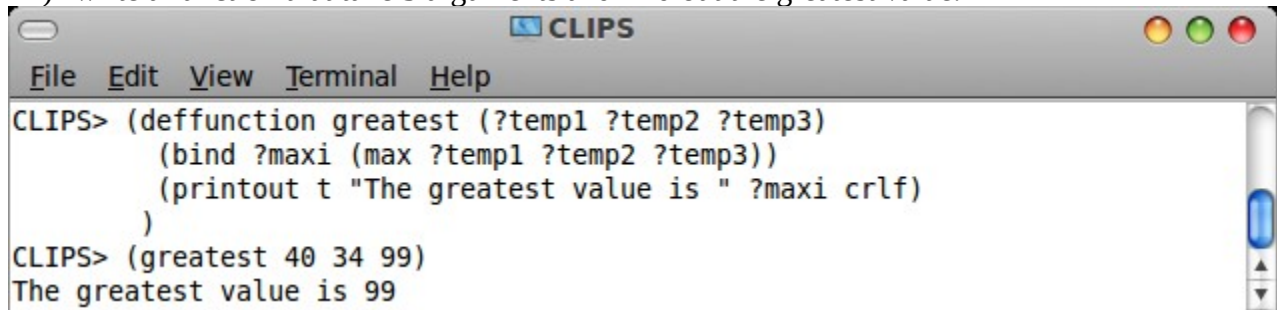
Reading from File:

A screenshot of a CLIPS terminal window. The window has a title bar with the text "CLIPS" and standard macOS window controls (red, yellow, green buttons). Below the title bar is a menu bar with the items "File", "Edit", "View", "Terminal", and "Help". The main area of the window contains a series of commands and their outputs. The commands are: "(open \"/home/louai/Desktop/clips/ReadMe.txt\" note \"w\")", "(printout note \"Hello to Expert System Basic Student Guide\")", "(close)", "(open \"/home/louai/Desktop/clips/ReadMe.txt\" note \"r\")", "(read note)", "(readline note)", and "(read note)". The outputs are: "TRUE", "TRUE", "Hello", and "EOF".

```
CLIPS> (open "/home/louai/Desktop/clips/ReadMe.txt" note "w")
TRUE
CLIPS> (printout note "Hello to Expert System Basic Student Guide")
CLIPS> (close)
TRUE
CLIPS> (open "/home/louai/Desktop/clips/ReadMe.txt" note "r")
TRUE
CLIPS> (read note)
Hello
CLIPS> (readline note)
" to Expert System Basic Student Guide"
CLIPS> (read note)
EOF
```

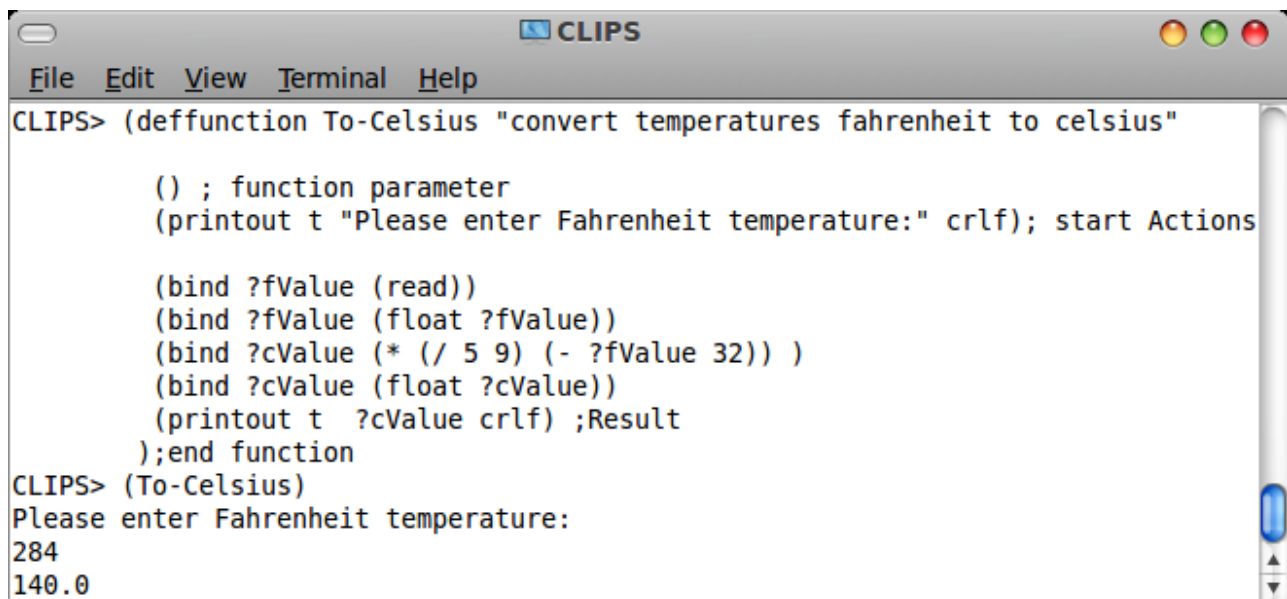
Exercise:

- 1) write a function that take 3 arguments and find out the greatest value.



```
CLIPS> (deffunction greatest (?temp1 ?temp2 ?temp3)
  (bind ?maxi (max ?temp1 ?temp2 ?temp3))
  (printout t "The greatest value is " ?maxi crlf)
)
CLIPS> (greatest 40 34 99)
The greatest value is 99
```

- 2) write a program to convert temperature from Fahrenheit to Celsius. Your program should accept a temperature from the user in Fahrenheit and print the corresponding temperature in Celsius.  
Hint: the formula for converting Fahrenheit ( f ) to Celsius ( c ) is:  $c = 5/9 (f - 32)$ .



```
CLIPS> (deffunction To-Celsius "convert temperatures fahrenheit to celsius"
  () ; function parameter
  (printout t "Please enter Fahrenheit temperature:" crlf); start Actions
  (bind ?fValue (read))
  (bind ?fValue (float ?fValue))
  (bind ?cValue (* (/ 5 9) (- ?fValue 32)) )
  (bind ?cValue (float ?cValue))
  (printout t ?cValue crlf) ;Result
);end function
CLIPS> (To-Celsius)
Please enter Fahrenheit temperature:
284
140.0
```



## Templates:

deftemplate construct like structure definition in other programming languages like C, C++ , Java...  
deftemplate gives the ability to abstract a structure of the facts by assigning a name to each field found within the facts.

Syntax:

```
(deftemplate <deftemplate-name> [<comment>]
  <slot-definition>*)

<slot-definition> ::= <single-slot-definition> | <multislot-definition>

<single-slot-definition> ::= (slot <slot-name> <template-attribute>*)

<multislot-definition> ::= (multislot <slot-name> <template-attribute>*)

<template-attribute> ::= <default-attribute> | <constraint-attribute>

<default-attribute> ::= (default ?DERIVE | ?NONE | <expression>*)
                      |(default-dynamic <expression>*)
```

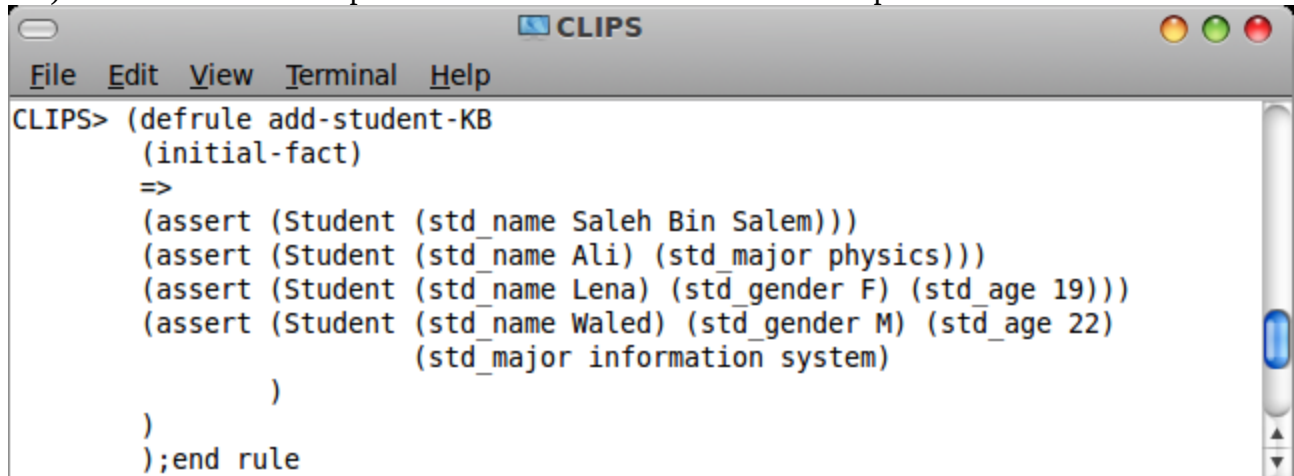
Example:

- 1) Suppose that we want to represent a knowledge base about students which has the following fields.

Slot-Type	Slot-name	Slot-default-Value
Multi slot	P_name	“ “
Single slot	P_gender	M
Single slot	P_age	23
Multi slot	P_major	Computer Science

```
CLIPS> (deftemplate Student "data related to Student"
  (multislot std_name (default " ") )
  (slot std_gender (default m ) )
  (slot std_age (default 23) )
  (multislot std_major (default undergraduate student))
);end Student template
CLIPS> (assert (Student (std_name Wael Bin Mishal) (std_age 24)) )
<Fact-0>
CLIPS> (facts)
f-0 (Student (std_name Wael Bin Mishal) (std_gender m) (std_age 24) (std_ma
or undergraduate student))
For a total of 1 fact.
```

- 2) From the same example above define a rule that insert a multiple student facts.

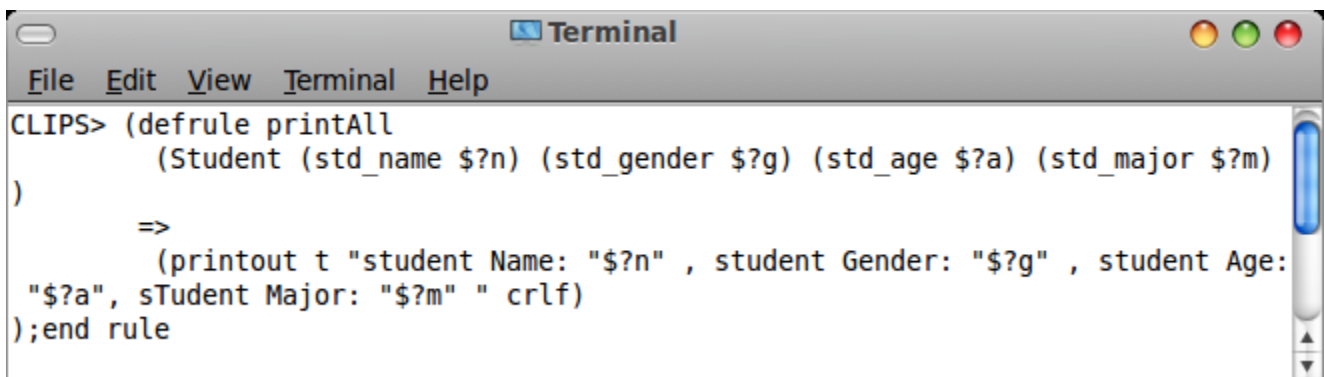


```
CLIPS> (defrule add-student-KB
  (initial-fact)
  =>
  (assert (Student (std_name Saleh Bin Salem)))
  (assert (Student (std_name Ali) (std_major physics)))
  (assert (Student (std_name Lena) (std_gender F) (std_age 19)))
  (assert (Student (std_name Waled) (std_gender M) (std_age 22)
    (std_major information system)
  )
  )
);end rule
```

After defining the rule execute the program so you can see the output like you have learned from previous section by typing


```
(reset)
(run)
(facts)
```

- 3) Form the same example above now define a rule that print all the facts related to the student



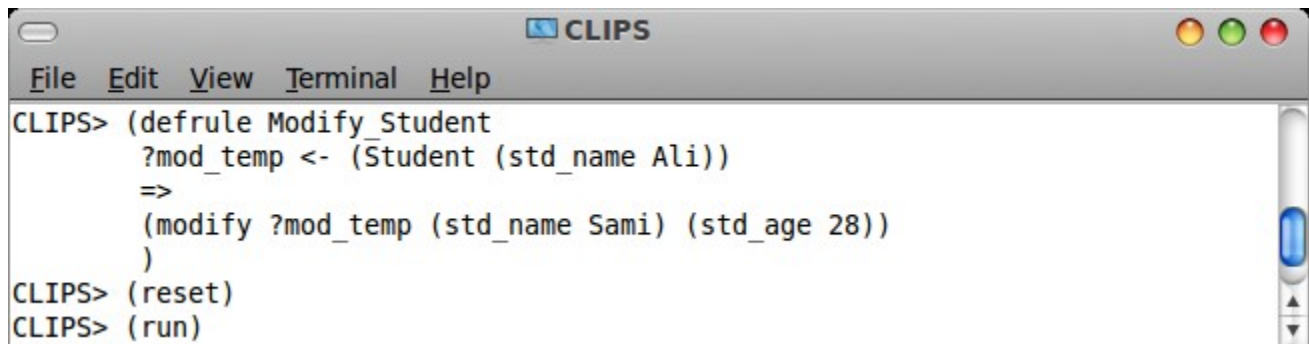
```
CLIPS> (defrule printAll
  (Student (std_name $?n) (std_gender $?g) (std_age $?a) (std_major $?m)
  )
  =>
  (printout t "student Name: "$?n" , student Gender: "$?g" , student Age:
    "$?a", sTudent Major: "$?m" " crlf)
);end rule
```

- 4) Suppose that we want to delete a facts in a template we will use (retract ) command, for example we want to delete all female from student template.



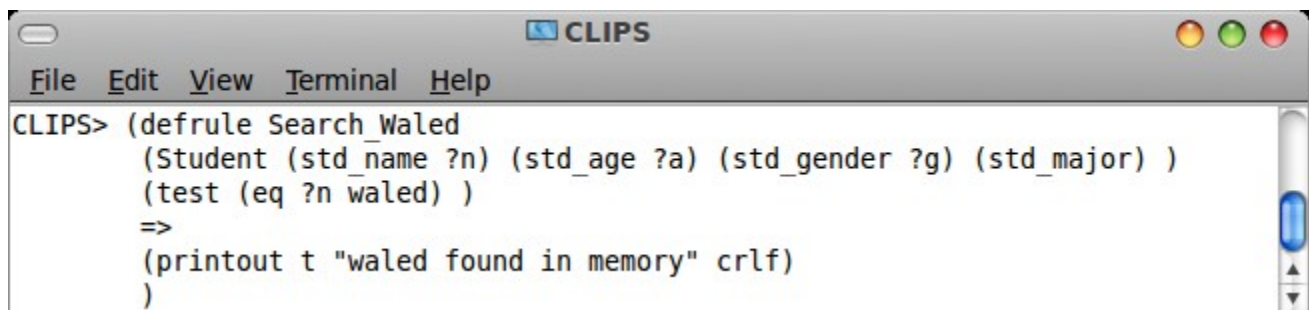
```
CLIPS> (defrule Delete_Female
  ?factID <-(Student (std_gender F))
  =>
  (retract ?factID)
  )
CLIPS> (reset)
CLIPS> (run)
```

- 5) Suppose that we have a knowledge that we want to modify we will use (modify) key word, for example modify the student name Ali to Sami and his age to 28.



```
CLIPS> (defrule Modify_Student
  ?mod_temp <- (Student (std_name Ali))
  =>
  (modify ?mod_temp (std_name Sami) (std_age 28))
)
CLIPS> (reset)
CLIPS> (run)
```

- 6) Finally we want to learn how to search for specific knowledge, we will use (test) keyword for that, Suppose that we want to search for student name waled.



```
CLIPS> (defrule Search_Waled
  (Student (std_name ?n) (std_age ?a) (std_gender ?g) (std_major) )
  (test (eq ?n waled) )
  =>
  (printout t "waled found in memory" crlf)
)
```

## Procedural Function in CLIPS

Procedural Function:

The following are functions which provide procedural programming capabilities as found in languages such as Pascal, C and Ada



IF condition:

Syntax:

```
(if <expression>
  then
    <action>*
  [else
    <action>*]
)
```

Example:

- 1) write a simple program that read integer form user and check wither if it's even or odd and assert facts about user input.

```
CLIPS> (deffunction EvenCheck ()
      (printout t "please enter integer value: " crlf)
      (bind ?tempInt (read))
      (bind ?tempInt (integer ?tempInt))
      (if (evenp ?tempInt)
        then
          (printout t ?tempInt " is even" crlf)
          (assert (even ?tempInt))
        else
          (printout t ?tempInt " is odd" crlf)
          (assert (odd ?tempInt))
      );endIf
    );endFunction
CLIPS> (EvenCheck)
please enter integer value:
83
83 is odd
<Fact-0>
```



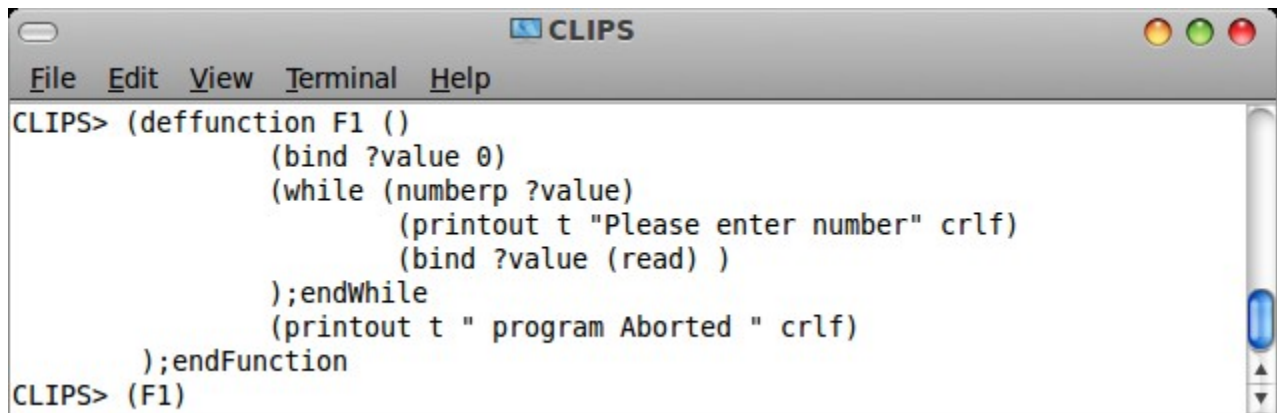
While Loop:

Syntax:

```
(while <expression> [do]
  <action>*)
)
```

Example:

- 1) write a simple program that read input form the user and check wither input is a number, if tis a number then let user input again otherwise abort from the program.



```
CLIPS> (deffunction F1 ()
      (bind ?value 0)
      (while (numberp ?value)
        (printout t "Please enter number" crlf)
        (bind ?value (read) )
      );endwhile
      (printout t " program Aborted " crlf)
    );endFunction
CLIPS> (F1)
```



For Loop:

Q: what is the different between a While loop and For loop ? (discussed this you TA)

Syntax:

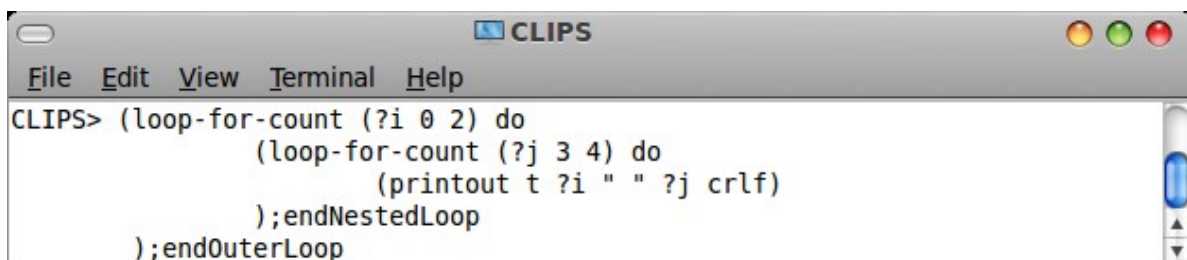
```
(loop-for-count <range-spec> [do] <action>*)

<range-spec> ::= <end-index> | (<loop-variable> <start-index> <end-index>) |
                  (<loop-variable> <end-index>)

<start-index> ::= <integer-expression>
<end-index> ::= <integer-expression>
```

Example:

- 1) Here an example to show instead for loop iteration



```
CLIPS> (loop-for-count (?i 0 2) do
      (loop-for-count (?j 3 4) do
        (printout t ?i " " ?j crlf)
      );endNestedLoop
    );endOuterLoop
```



## Foreach Loop:

Syntax:

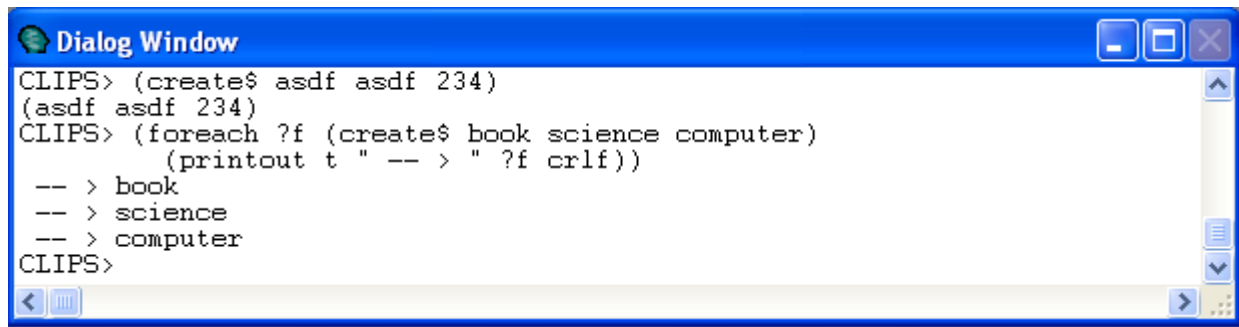
```
(foreach <list-variable> <multifield-expression>
  <expression>*)
)
```

Example:

The example below show:

(create\$ ) → to create multiple field

using foreach to print out each field of chine.



```
Dialog Window
CLIPS> (create$ asdf asdf 234)
(asdf asdf 234)
CLIPS> (foreach ?f (create$ book science computer)
  (printout t " -- > " ?f crlf))
-- > book
-- > science
-- > computer
CLIPS>
```

# Inference Engine Technique

Backward chaining (goal driven) : the inference engine works backward from a conclusion to be proven to determine if the facts in working memory prove the truth of the conclusion.

Example:

Rule-base

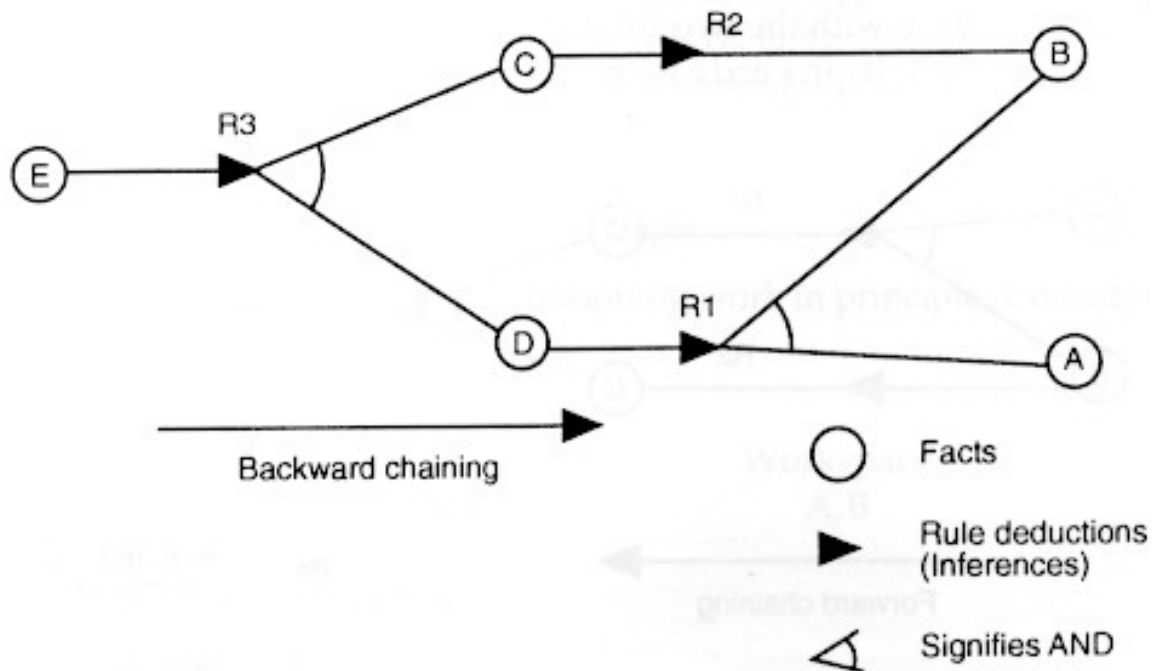
R1: IF A AND B THEN D

R2: IF B THEN C

R3: IF C AND D THEN E

Working Memory

A,B



## Expert system for Car diagnosis.

- Rule 1: IF the engine is not getting gas  
AND the engine will turn over  
THEN the problem is engine need gas.
- Rule 2: IF the engine does turn over  
AND the engine has gas.  
THEN the problem is need to contact with provider.
- Rule 3: IF the engine does not turn over  
AND the lights do not come on  
THEN the problem is battery or cables.
- Rule 4: IF there engine is not trun over  
AND the light is come on AND solenoid is working AND terminal is clean  
THEN the starter need to replace.
- Rule 5: IF engine does not turn over  
AND lights come on AND solenoid is working  
AND terminal is not clean  
THEN clean termianl.
- Rule 6: IF engine does not turn over  
AND lights come on AND solenoid is not working  
AND solenoid fuse is not working  
THEN fuse need to be replace.
- Rule 7: IF engine does not turn over  
AND lights come on AND solenoid is not working  
AND solenoid fuse is working  
THEN solenoid need to be replace.

Solution:

.....  
.....

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;; Car Diagnosis Example
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(deftemplate question
  "A question the application may ask"
  (slot text)      ;; The question itself
  (slot ident))    ;; The "name" of the question

(deftemplate answer
  (slot ident)
  (slot text))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Knowledge base
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(deffacts question-data
  (question
    (ident starter)
    (text "Does the car's starter turningg (yes or no)? "))
  (question
    (ident petrol)
    (text "Do you have any petrol (yes or no)? "))
  (question
    (ident light)
    (text "Are the lights working (yes or no)? "))
  (question
    (ident solenoid)
    (text "Does the solenoid click (yes or no)? "))
  (question
    (ident terminals)
    (text "Are the terminals clean (yes or no)? "))
  (question
    (ident solenoid-fuse)
    (text "Are the fuse working (yes or no)? "))
  (ask starter)
  (ask petrol)
  (ask light)
  (ask solenoid)
  (ask terminals)
  (ask solenoid-fuse)
)

```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;  inference Engine
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
(defrule r1
  (answer (ident starter) (text yes) )
  (answer (ident petrol) (text no) )
```

```
=>
```

```
(assert (buy some petrol!))
(printout t "buy some petrol!" crlf)
```

```
)
```

```
(defrule r2
  (answer (ident starter) (text yes) )
  (answer (ident petrol) (text yes) )
```

```
=>
```

```
(assert (call provider))
(printout t "call provider" crlf)
```

```
)
```

```
(defrule r3
  (answer (ident starter) (text no) )
  (answer (ident light) (text no) )
```

```
=>
```

```
(assert (charge battery))
(printout t "charge battery" crlf)
```

```
)
```

```
(defrule r4
  (answer (ident starter) (text no) )
  (answer (ident light) (text yes) )
  (answer (ident solenoid) (text yes) )
  (answer (ident terminals) (text yes) )
```

```
=>
```

```
(assert (replace starter))
(printout t "replace starter" crlf)
```

```
)
```

```
(defrule r5
  (answer (ident starter) (text no) )
  (answer (ident light) (text yes) )
  (answer (ident solenoid) (text yes) )
  (answer (ident terminals) (text no) )

  =>
  (assert (clean terminals))
  (printout t "clean terminals" crlf)
)

(defrule r6
  (answer (ident starter) (text no) )
  (answer (ident light) (text yes) )
  (answer (ident solenoid) (text no) )
  (answer (ident solenoid-fuse) (text no) )

  =>
  (assert (replace fuse))
  (printout t "replace fuse" crlf)
)

(defrule r7
  (answer (ident starter) (text no) )
  (answer (ident light) (text yes) )
  (answer (ident solenoid) (text no) )
  (answer (ident solenoid-fuse) (text yes) )
  =>
  (assert (replace solenoid))
  (printout t "replace solenoid" crlf)
)
```

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; backward chaining rule
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defrule supply-answers
  (answer (ident ?id))
  (not (answer (ident ?id)))
  (not (ask id?))
  =>
  (assert (ask ?id)))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Define Function
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; Ask user function
(defun ask-user (?question)
  "Ask a question, and return the answer"
  (bind ?answer "")
  (printout t ?question " ")
  (bind ?answer (read))
  (return ?answer))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; Ask by ID Rule
(defrule ask-question-by-id
  "ask user by id"
  (question (ident ?id) (text ?text))
  (not (answer (ident ?id)))
  ?ask <- (ask ?id)
  =>
  (retract ?ask)
  (bind ?answer (ask-user ?text))
  (assert (answer (ident ?id) (text ?answer)))
  )

;;;;;;;;;;;;;;;; END ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```

# CLIPS Integration with other Programming Languages

CLIPS could be integrated with any other languages you can go to CLIPS web site to see the available language that could have integration with CLIPS, this is tutorial about integrating CLIPS with Java and C++.

Lab Requirement for this part of document:

- 1) Install JDK ( Java Development Kit )
- 2) Install any Java IDE (Integrated Development Environment )
- 3) Install JESS from the following website <http://www.jessrules.com/>

Skills Requirement:

Ability to understand API ( Application programming Interface):

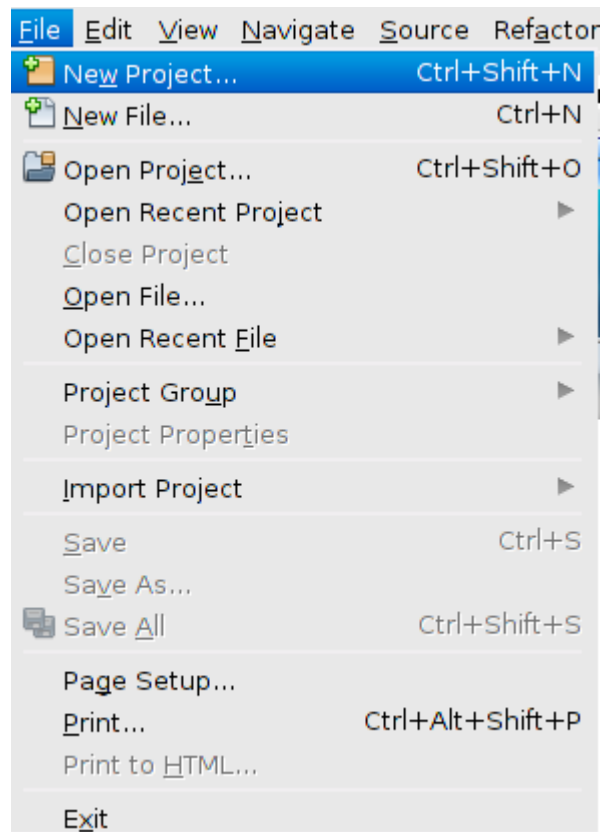
Java API use the following link: <http://java.sun.com/reference/api/>

JESS API use the following link: <http://www.jessrules.com/jess/docs/index.shtml>

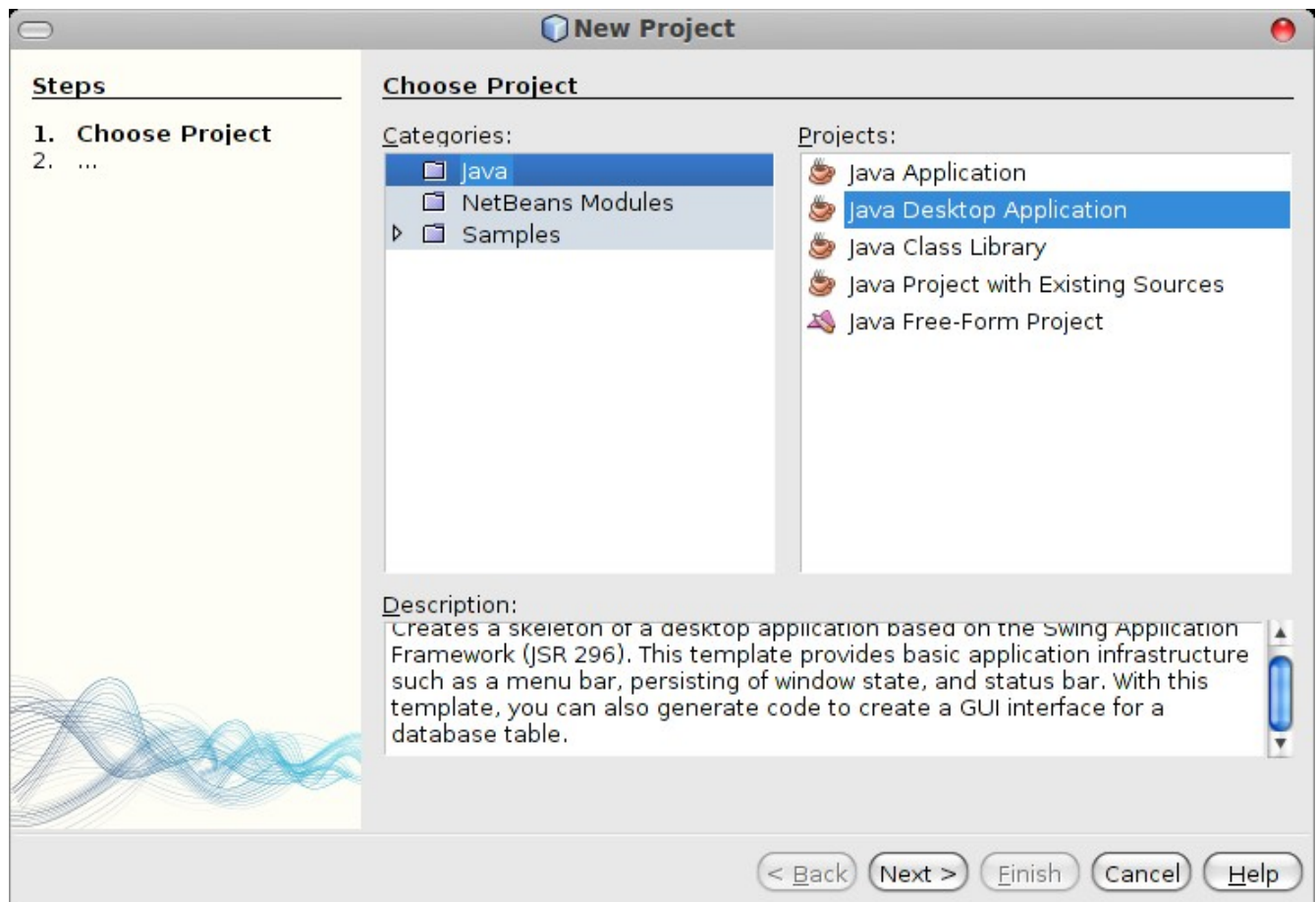


In this Tutorial I used NetBeans IDE , all concept provided also could be applied for any another Java IDE

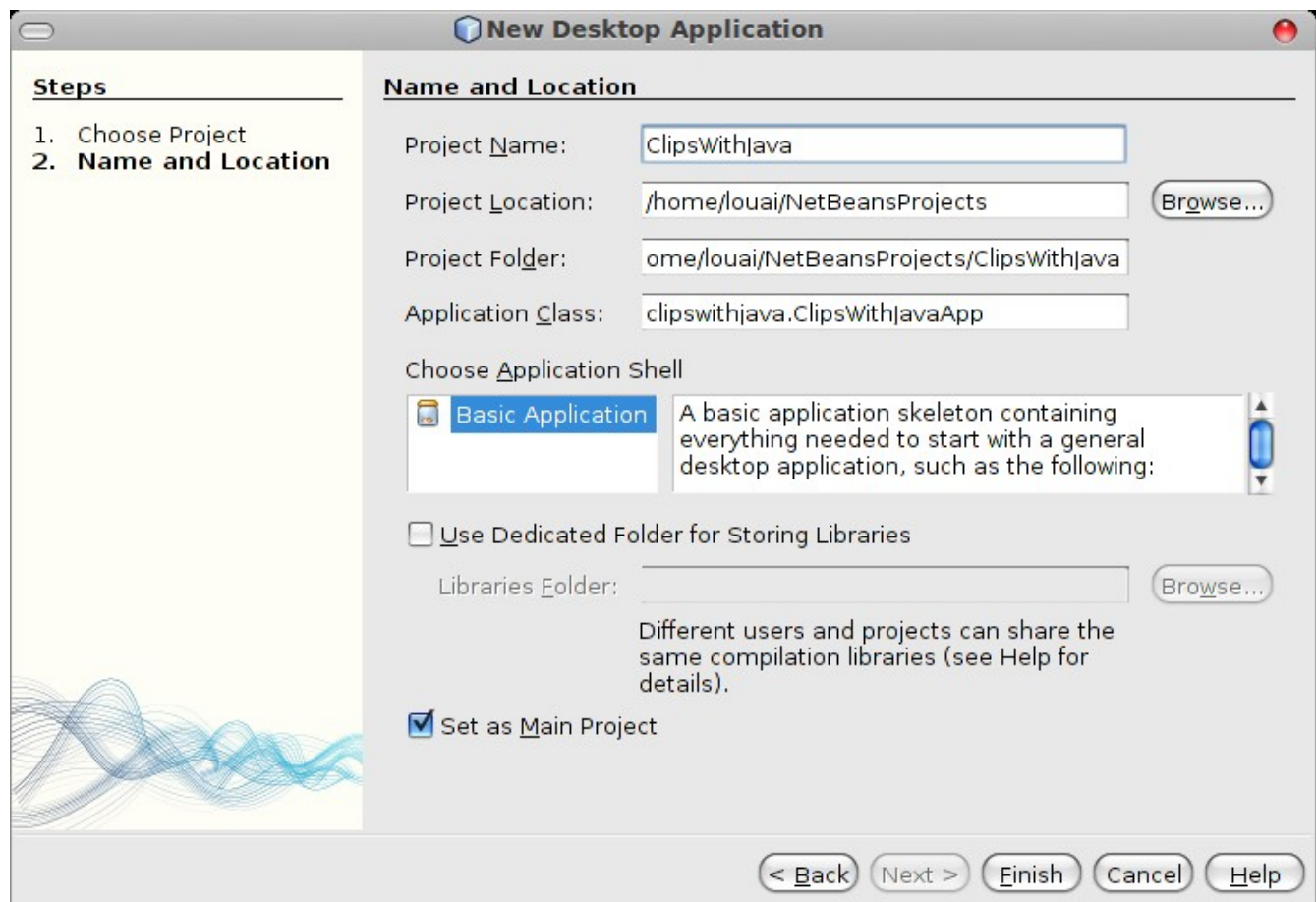
- 1) Create a project.



Chose the Project type among projects :



Enter a name of the project :



**New Desktop Application**

**Steps**

1. Choose Project
- 2. Name and Location**

**Name and Location**


Project Name:

Project Location:  [Browse...](#)

Project Folder:

Application Class:

Choose Application Shell

 <b>Basic Application</b>	A basic application skeleton containing everything needed to start with a general desktop application, such as the following:
--	---

☐ Use Dedicated Folder for Storing Libraries

Libraries Folder:  [Browse...](#)

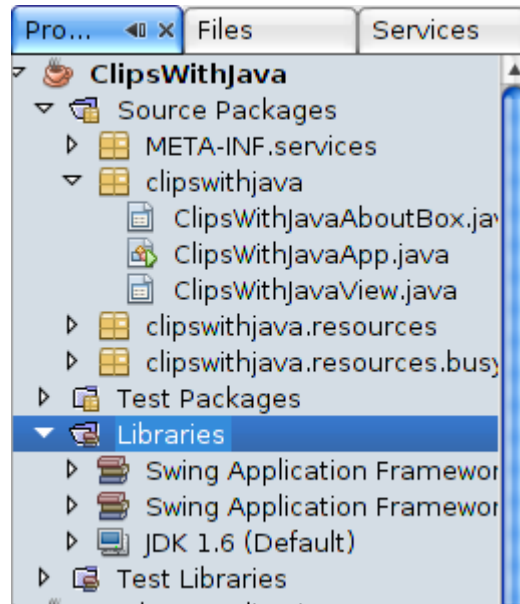
Different users and projects can share the same compilation libraries (see Help for details).

☒ Set as Main Project

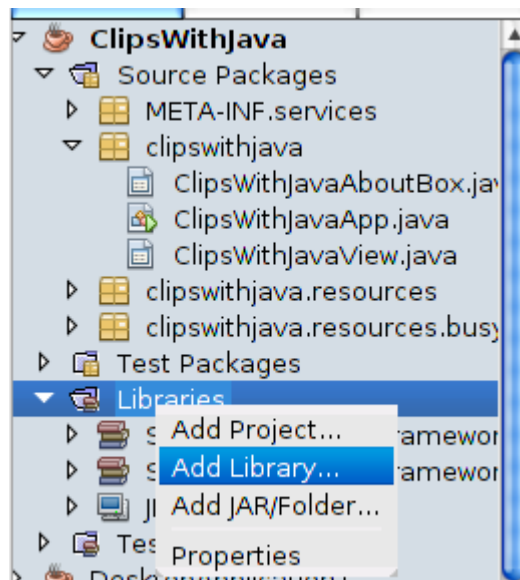
< [Back](#)   [Next](#) >   [Finish](#)   [Cancel](#)   [Help](#)

## 2) Include JESS library to your project

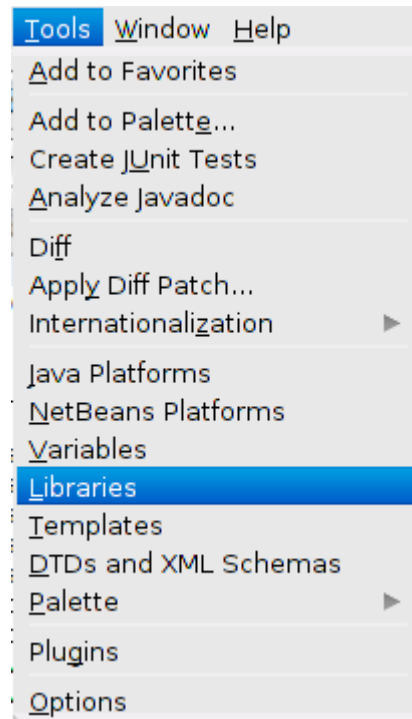
There is First way to include library through project Folder go to library like figure below.

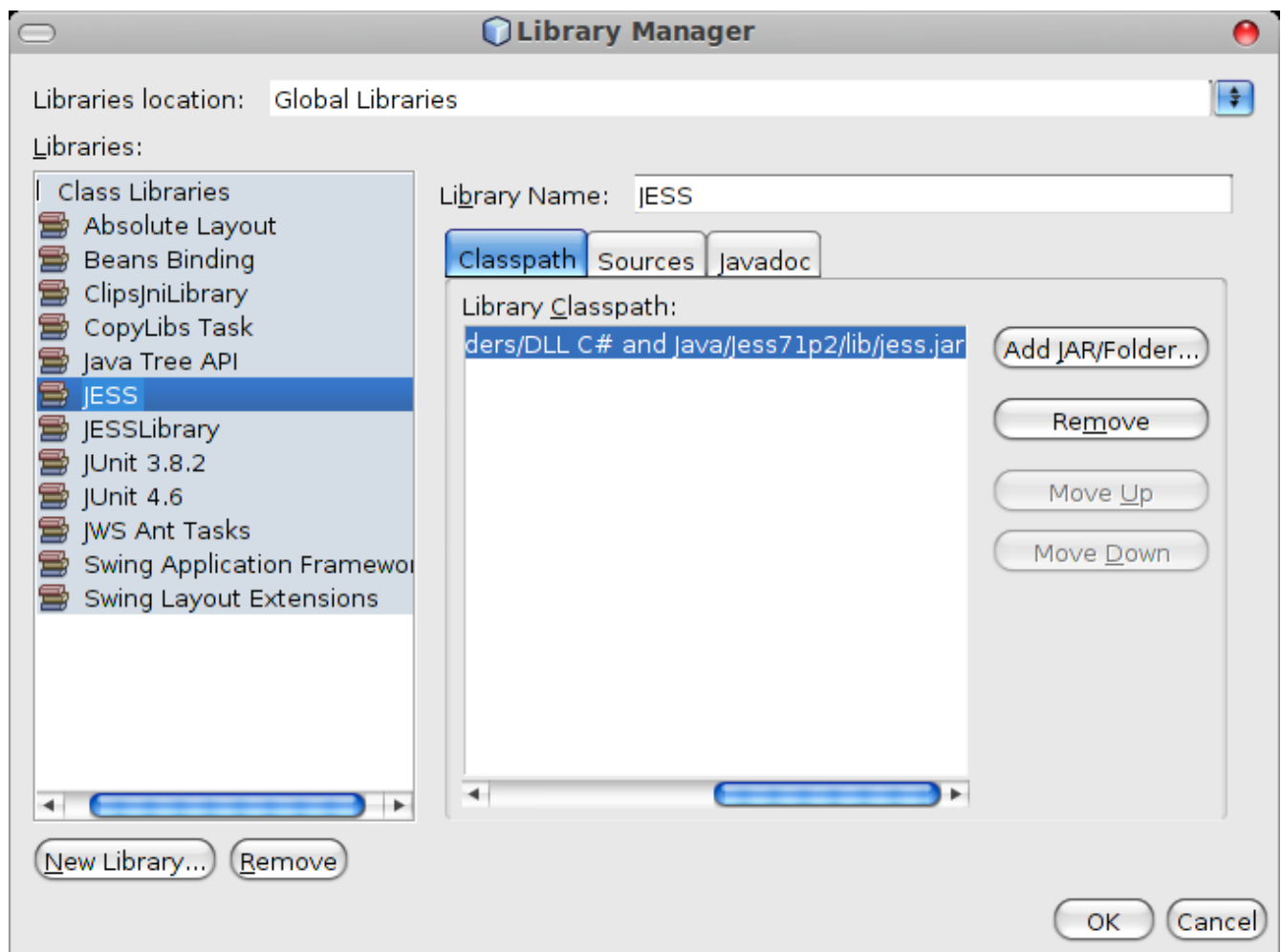


Click on Right-Mouse and select form the pub pap menu Add Library



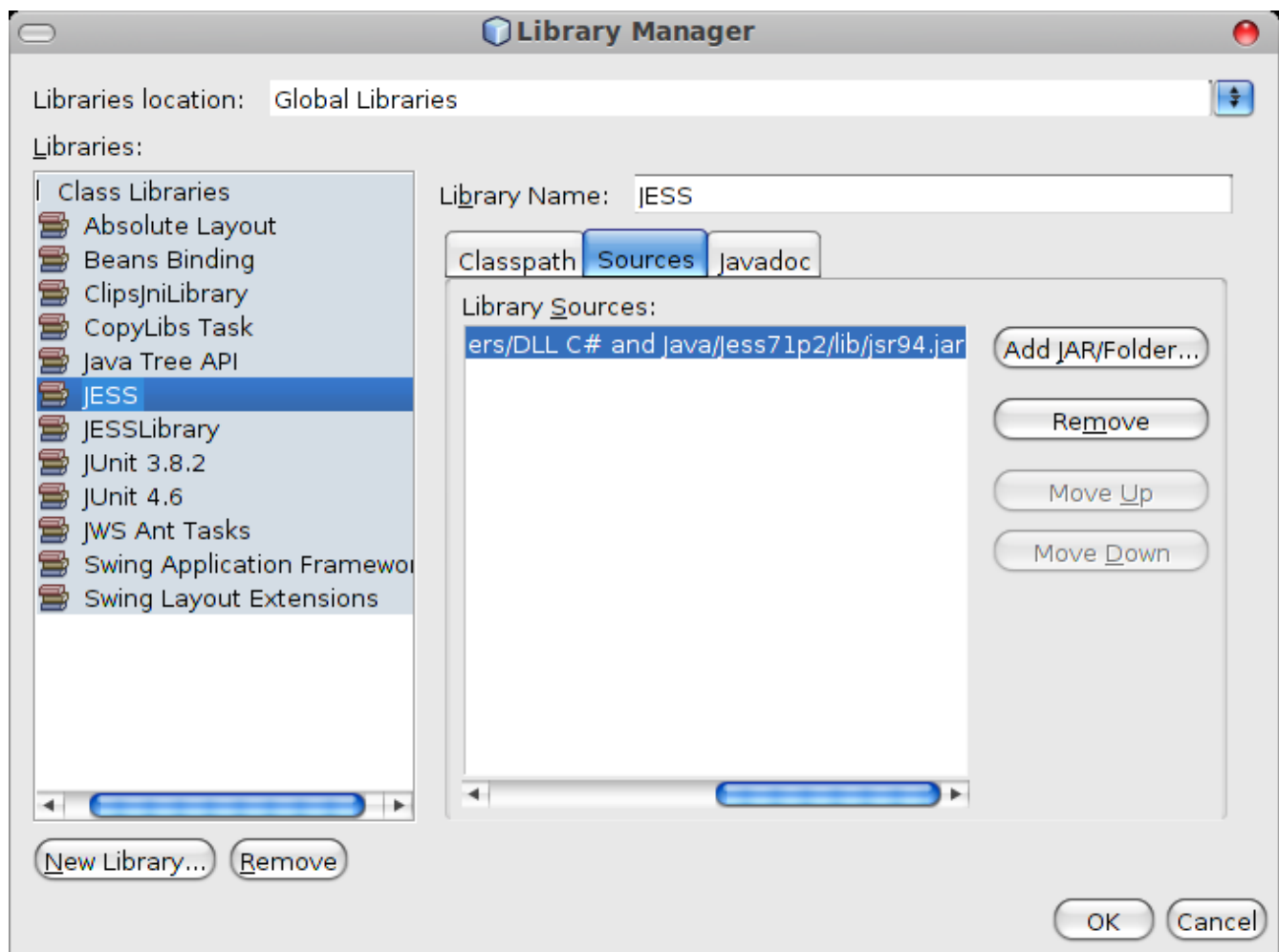
Another way to include a Library throw Tool:





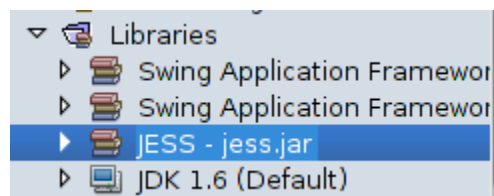
A pub pap form will be shown :

- 1- Click on new Library
- 2- Enter New Library name ( JESS )
- 3- Press Ok.
- 4- a new defined library will be in the left list.
- 5- Add jess.jar to ClassPath
- 6- Add jsr94.jar to Sources
- 7- press Ok.



### 3) Simple application.

After adding the Library, now you have to include it to your project see the below figure



now import the Library by the following code

```
import jess.*;
```

```
import jess.*;

/**
 *
 * @author louai
 */
public class Main {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        Rete r = new Rete();
        String command = "(deffunction foo(?x) "+
            "(return (+ ?x 1))"
            +")";

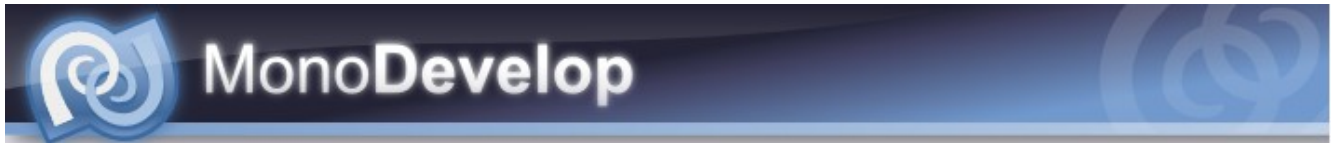
        try {
            r.eval(command);
            Value v = r.eval("(foo 3)");
            System.out.println(v.intValue(r.getGlobalContext()));
        } catch (JessException ex) {
            Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex)
        }
    }
}
```

Second Example:

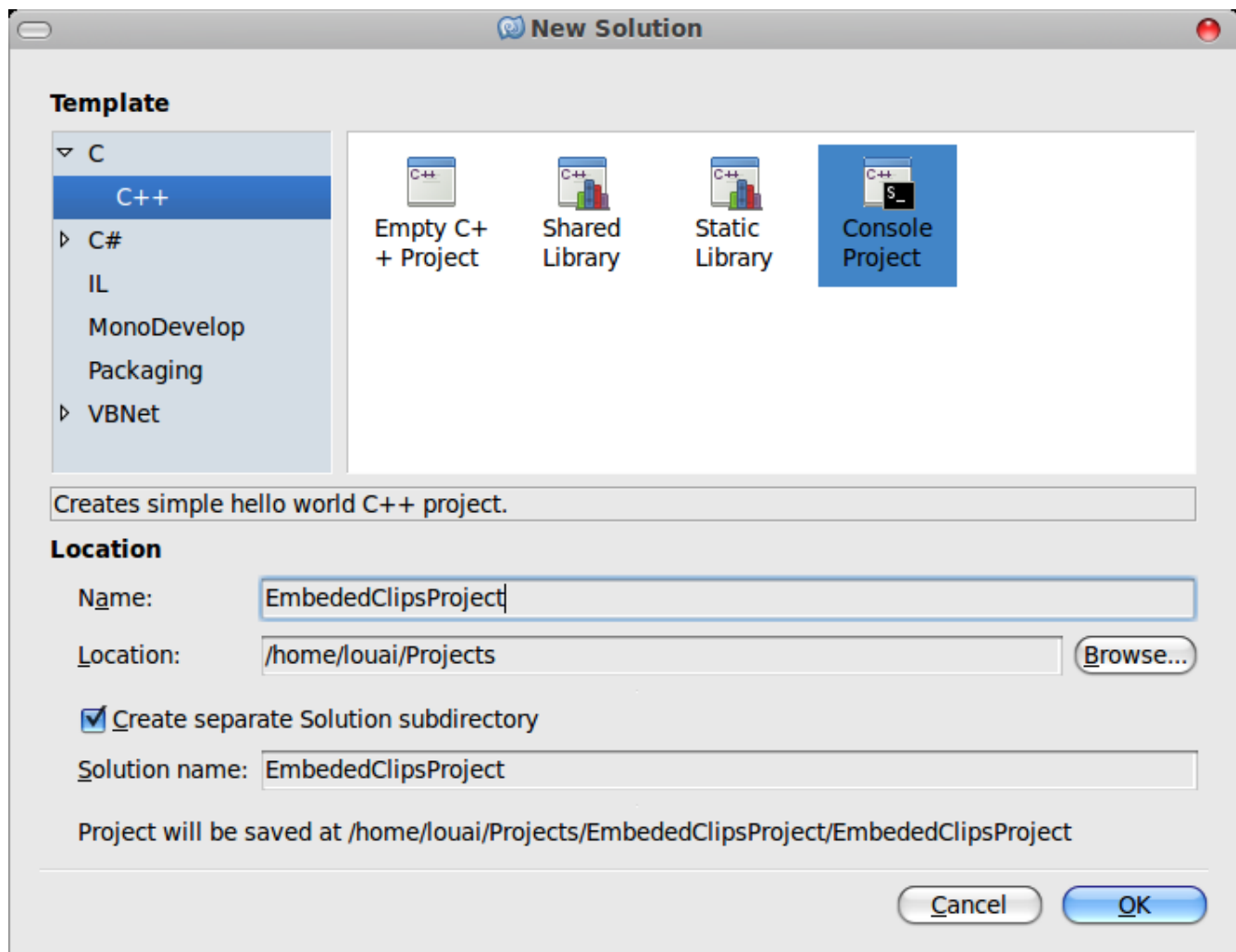
Integrate CLIPS with c++ application,

Step 1. Download clips binding library from CLIPS website, Here I used the library form <http://www.ghg.net/clips/download/source/> .

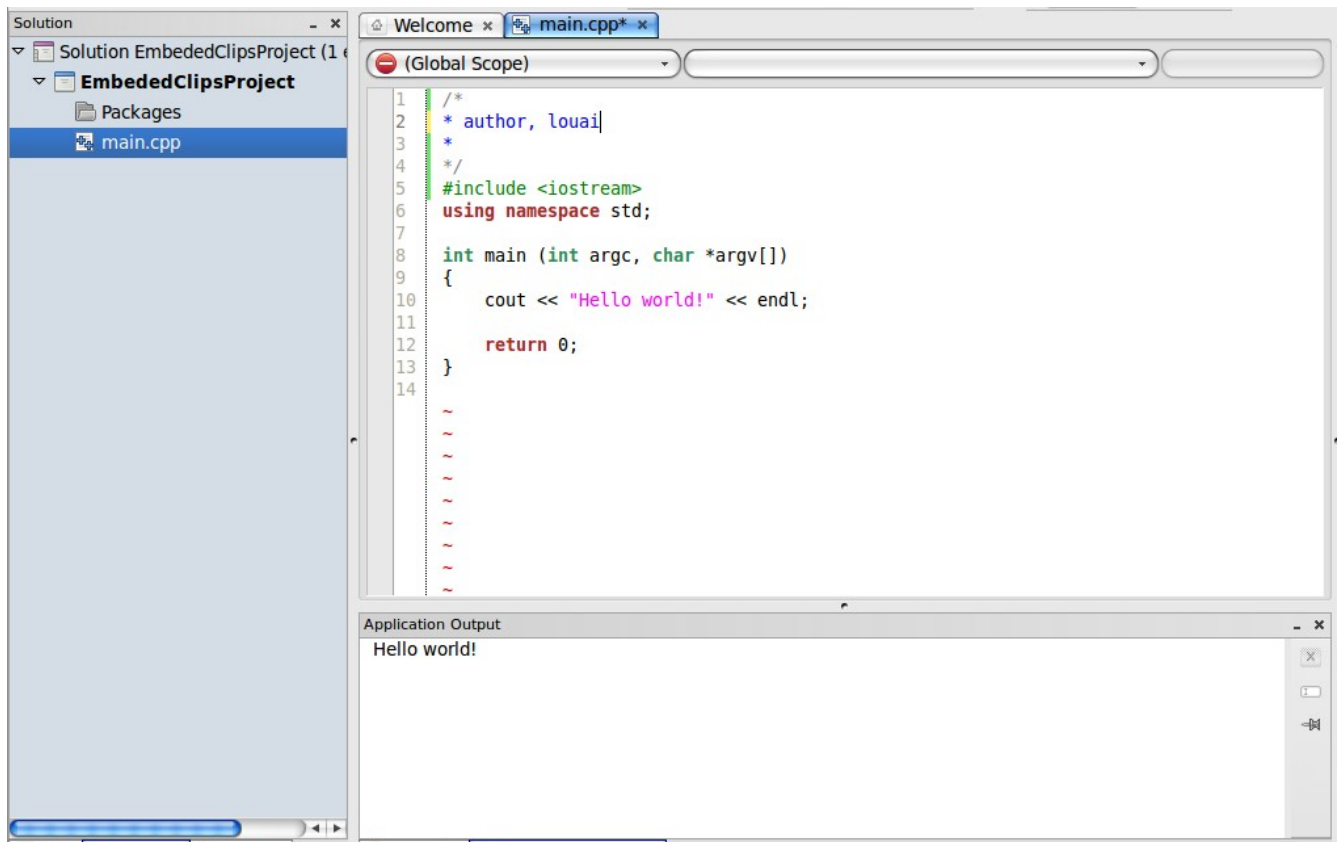
Step 2. Use any IDE for writing C++ , such Visual Studio , Eclipse , NetBeans or MonoDevelop  
Here I will use MonoDevelop IDE.



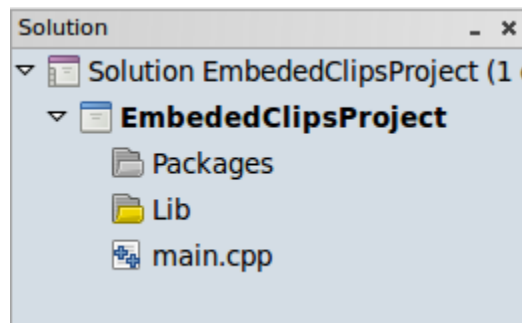
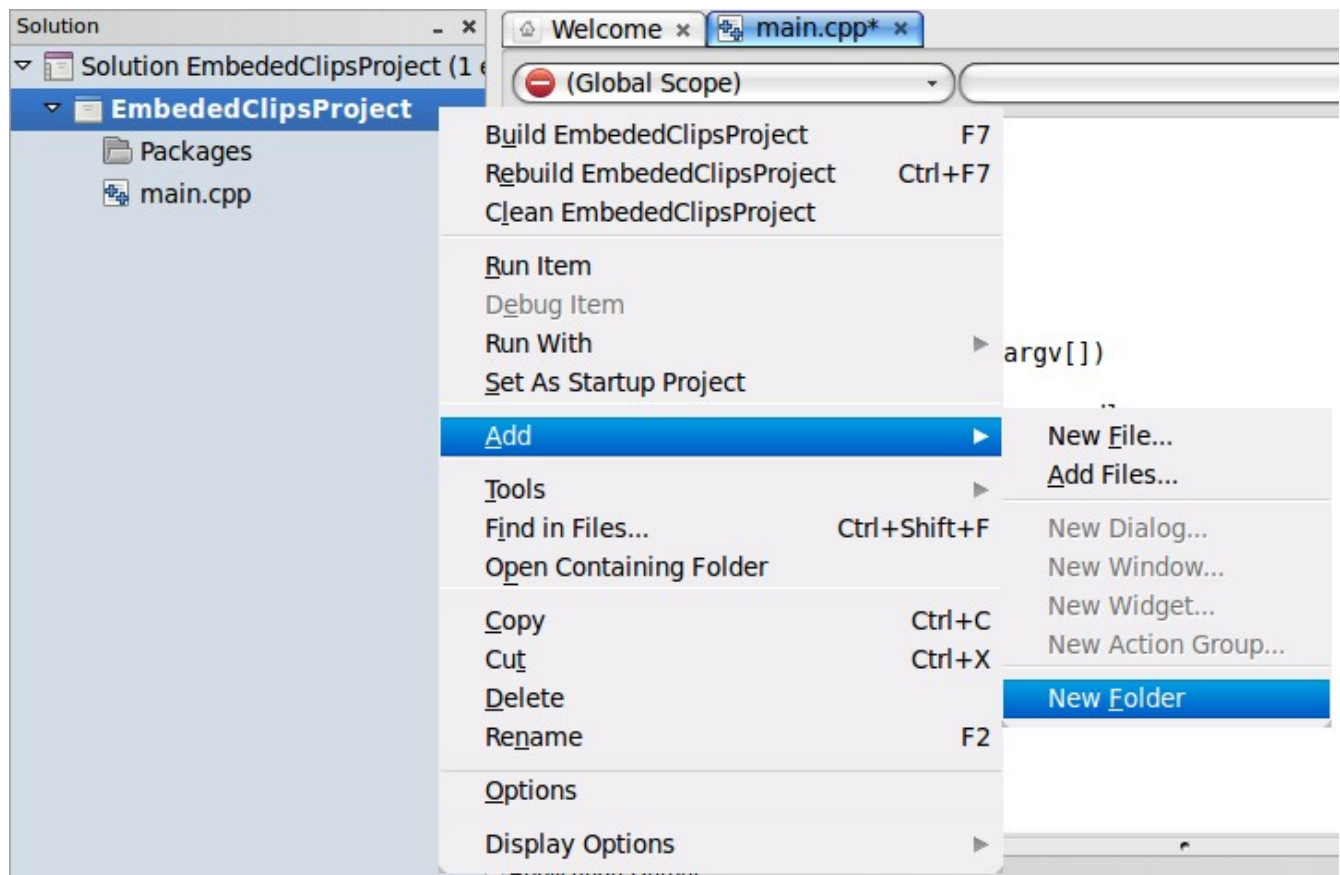
From menu bar chose File → new → solution a wizard will appear to create a new project.



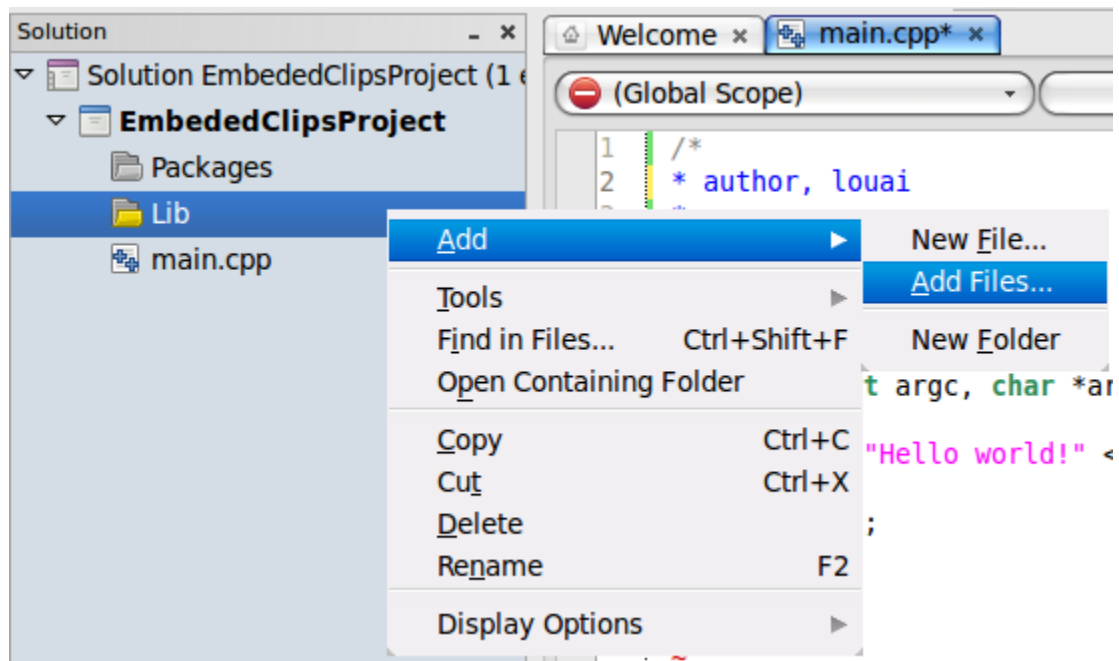
Then complete the wizard. Build & run your project to test it (Hello World!).

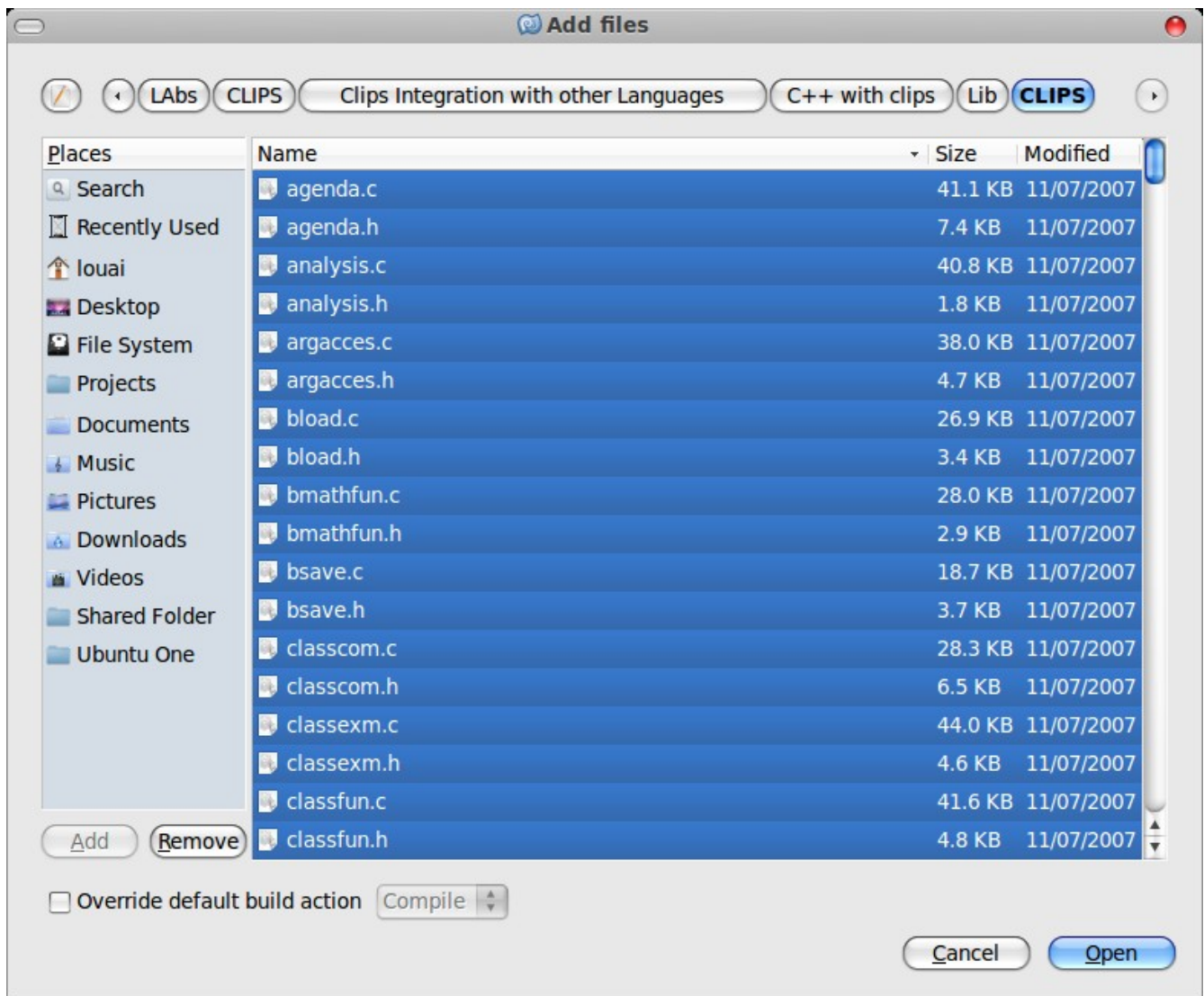


Step3. Create a new folder in Solution call it Lib for library.

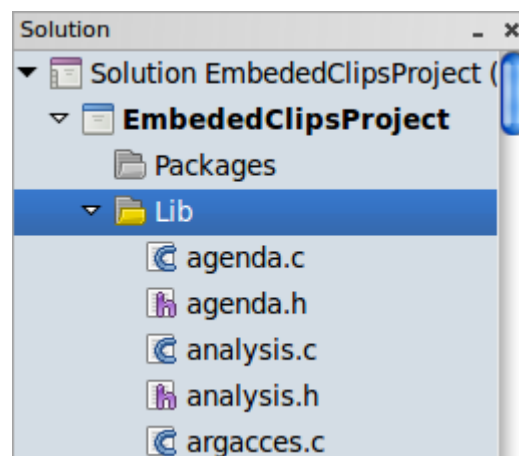


Step 3. Extract clips library source and copy the extracted library in to Lib folder in your project.

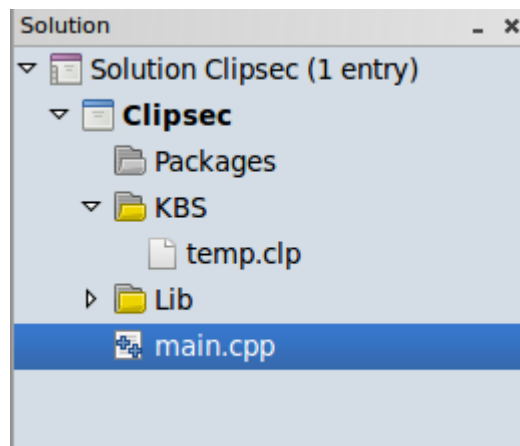




Step 4. Now delete a main.c in Lib folder in your project because you already created your main function in C++.



Step 5. Create a new folder in the same project call it KBS and create new file in it name it temp.clp

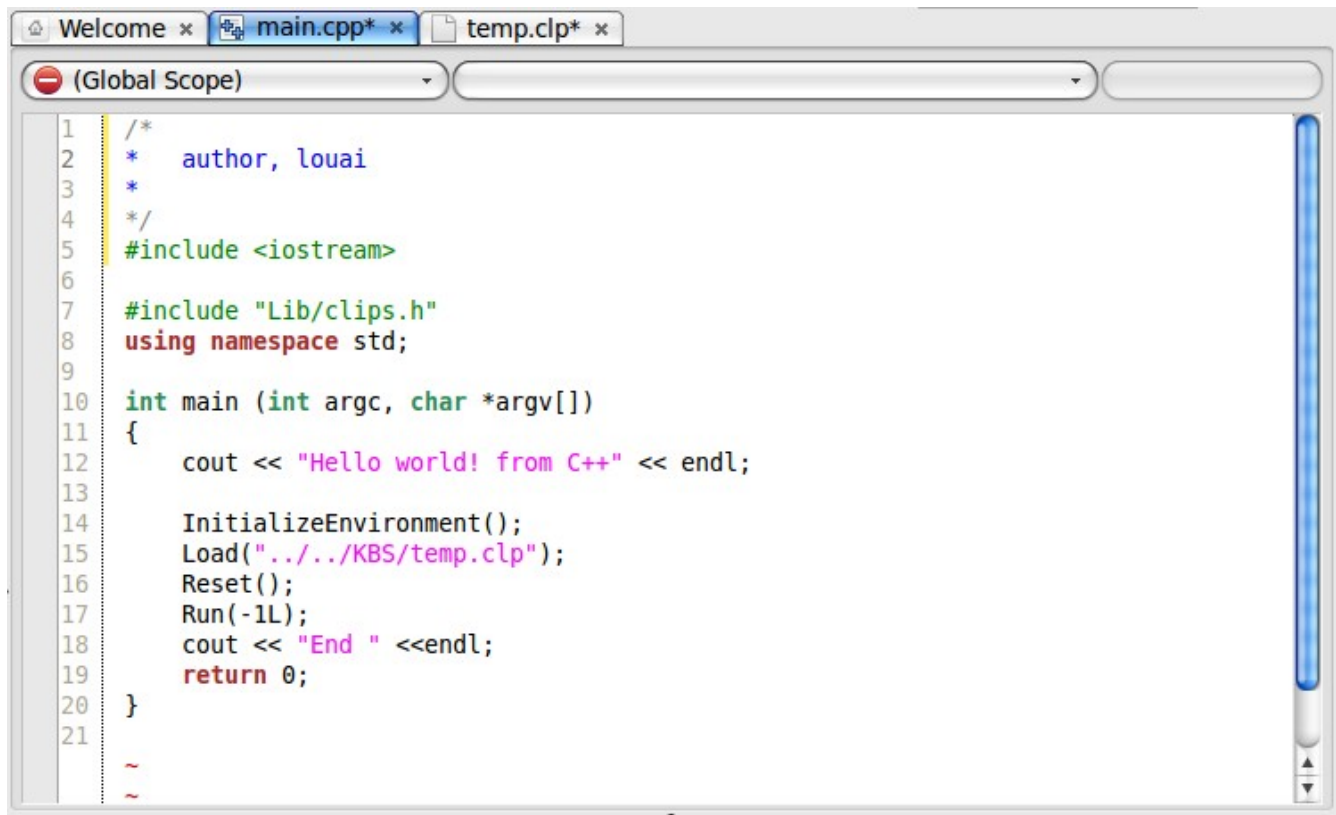


choose to open it in text editor and enter in this file your clips code

A screenshot of the Visual Studio text editor with three tabs: 'Welcome', 'main.cpp', and 'temp.clp\*'. The 'temp.clp' tab is active. The code in the editor is as follows:

```
1 (defacts asdf
2   (hello man)
3 );endfacts
4
5 (defrule r1
6   =>
7   (printout t "Hello to Clips World integrated with C++| " crlf)
8 );end r1
9
~
~
~
~
~
~
~
~
~
~
~
```

Step 6. Now in your main include “`Lib/clips.h`” and modify your main to like the following



```
1  /*
2  *   author, louai
3  *
4  */
5  #include <iostream>
6
7  #include "Lib/clips.h"
8  using namespace std;
9
10 int main (int argc, char *argv[])
11 {
12     cout << "Hello world! from C++" << endl;
13
14     InitializeEnvironment();
15     Load("../KBS/temp.clp");
16     Reset();
17     Run(-1L);
18     cout << "End " << endl;
19     return 0;
20 }
21
```

Finally build your project and run, the output should look like this.

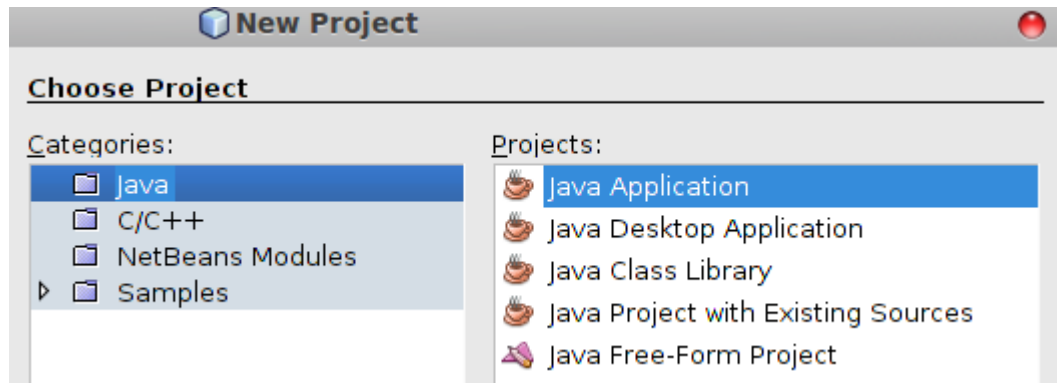


```
Application Output
Hello world! from C++
Hello to Clips World integrated with C++
End
```

# Expert system for Car diagnosis with Java Application

From previous tutorial below steps has been shown.

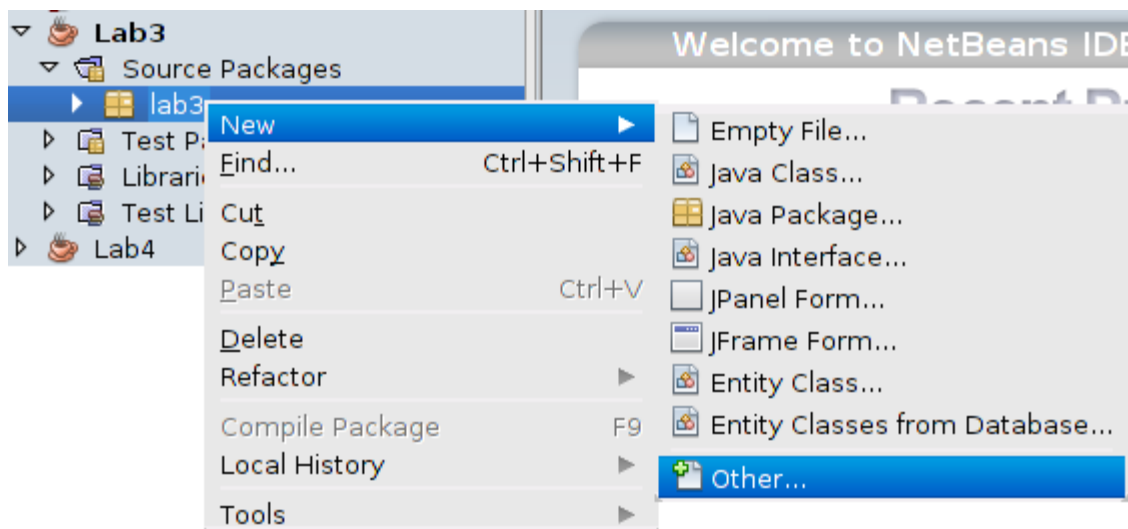
- 1) Create new project.



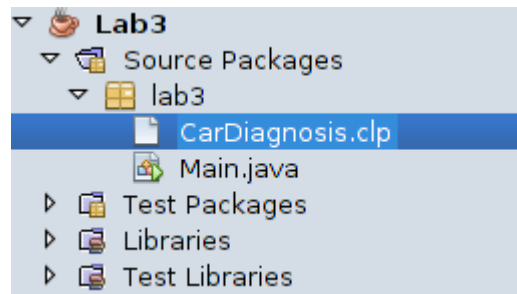
- 2) Add JESS library

Now we will do the following:

- 3) Add .clp file to the source package.



- 4) Name a file CarDiagnosis.clp.



- 5) Open clp File by double click and then write the CLIPS code from Car Diagnosis Example.

```
1  ;;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
2  ;;:::::::::      Car Diagnosis Example
3  ;;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
4
5  (deftemplate question
6    "A question the application may ask"
7    (slot text)      ;; The question itself
8    (slot ident))    ;; The "name" of the question
9
10 (deftemplate answer
11   (slot ident)
12   (slot text))
13 ;;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
14 ;; backward chaining answer (my goal)
15 (do-backward-chaining answer)
16 ;;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
17 (deffacts question-data
18   (question (ident starter) (text "Does the car's starter turningg (yes o
19   (question (ident petrol) (text "Do you have any petrol (yes or no)? "))
20   (question (ident light) (text "Are the lights working (yes or no)? "))
21   (question (ident solenoid) (text "Does the solenoid click (yes or no)?
22   (question (ident terminals) (text "Are the terminals clean (yes or no)?
23   (question (ident solenoid-fuse) (text "Are the terminals clean (yes or
24   (ask starter)
```

6) Create a function in Main.java to read text from .clp file.

```
public static String ReadFile(File pFile) {
    FileReader reader = null;
    String content = "";

    try {
        int i = 0;
        reader = new FileReader(pFile);
        while (i != -1) {
            i = reader.read();
            content += (char) i;
        }
    }
    catch (IOException ex) {
        Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
    }
    finally {
        try {
            reader.close();
        } catch (IOException ex) {
            Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    return content;
}

//end Function Read Text from file.
```

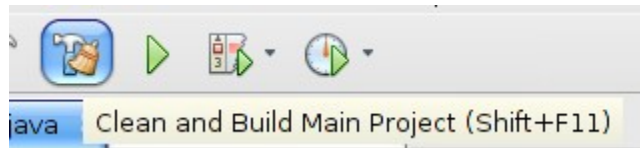
7) Inside Main Thread Write the following code

```
public static void main(String[] args) {
    // TODO code application logic here
    String path = "../Lab3/build/classes/lab3/CarDiagnosis.clp";
    Rete engine = new Rete();
    File file = new File(path);
    String command = ReadFile(file);
    try {
        engine.executeCommand(command);
        engine.reset();
        engine.reset();
        engine.run();
        Value v = engine.eval(command);
        System.out.println(v.intValue(engine.getGlobalContext()));
    } catch (JessException exf) {
        Logger.getLogger(Main.class.getName()).log(Level.SEVERE,
            null, ex);
    } finally {
        System.out.println("End Application");
    }

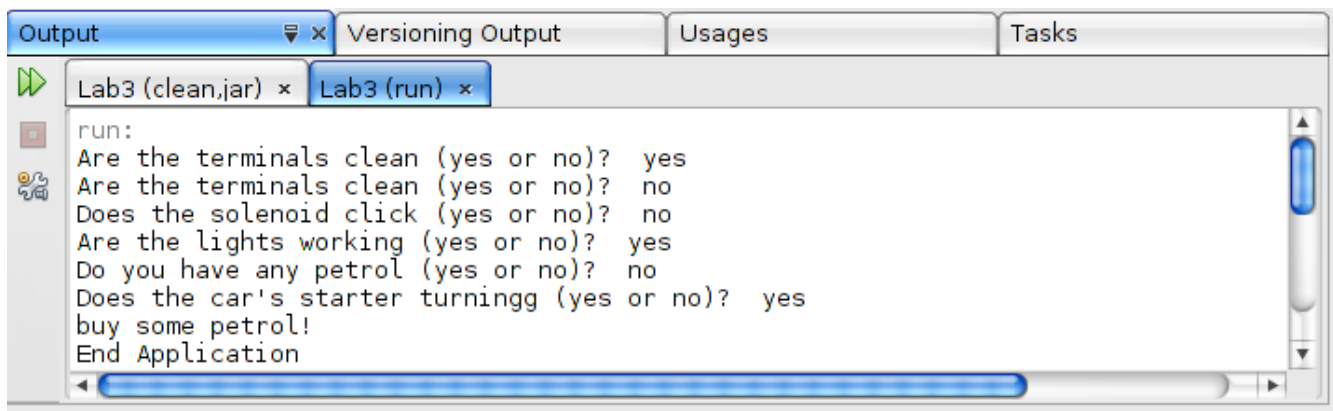
}

} //End main Thread.
```

8) Finally Build and Run the project.



9) Output in Terminal should look like this.



# Expert system for Car diagnosis with Java Desktop Application

Finally we want to create Desktop application for the same example above, with the following requirement:

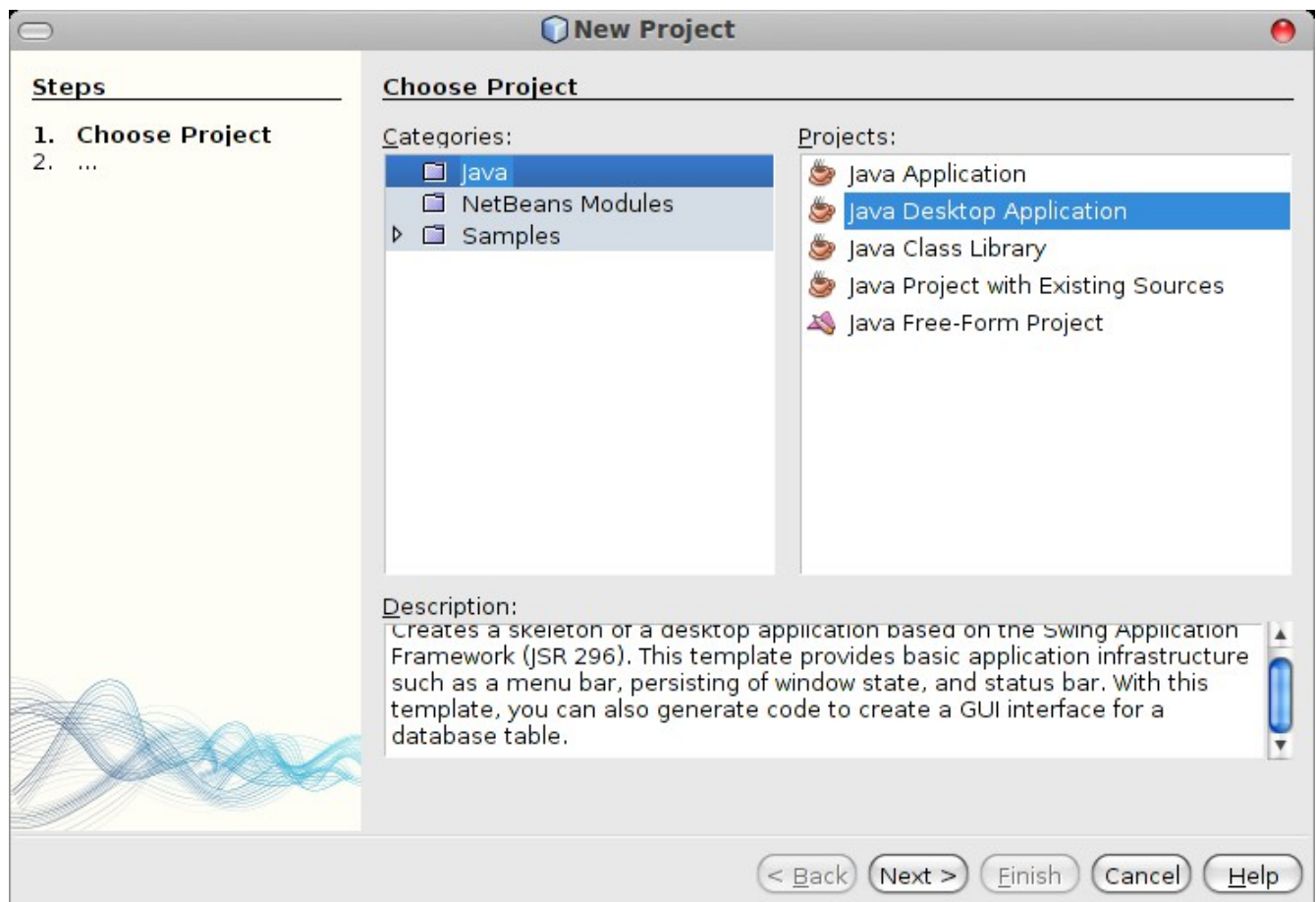
- 1- Create a Desktop Application that load Diagnosis file
- 2- make a project dynamic to be fit with any Diagnosis file as possible.

Download the Project form following site:

<https://sites.google.com/site/orabilouai/my-library/expert-system-clips/>

Step by Step Tutorial:

- (1) Create a Desktop Project



- (2) Add JESS library, use the previous tutorial if you still do not know how to import library to your project.

(3) Add .clp file to source package, then modify it.

Hint:

- user the previous tutorial if you still do not know How to include clp file to source package.
- Notice that we delete some rule that interact with user terminal.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;; Car Diagnosis Example
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(deftemplate question
  "A question the application may ask"
  (slot text)      ;; The question itself
  (slot ident))    ;; The "name" of the question

(deftemplate answer
  (slot ident)
  (slot text))

(deftemplate advice
  (slot result)
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; backward chaining answer (my goal)
(do-backward-chaining answer)
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(deffacts question-data
  (question
    (ident starter)
    (text "Does the car's starter turningg (yes or no)? "))
  (question
    (ident petrol)
    (text "Do you have any petrol (yes or no)? "))
  (question
    (ident light)
    (text "Are the lights working (yes or no)? "))
  (question
    (ident solenoid)
    (text "Does the solenoid click (yes or no)? "))
  (question
    (ident terminals)
    (text "Are the terminals clean (yes or no)? "))
  (question
    (ident fuse)
    (text "Are the fuse working (yes or no)? "))
)
```

```

;-----
(defrule r1
  (answer (ident starter) (text yes) )
  (answer (ident petrol) (text no) )

  =>
  (assert (advice (result "buy some petrol!")))
  (printout t "buy some petrol!" crlf)
)

(defrule r2
  (answer (ident starter) (text yes) )
  (answer (ident petrol) (text yes) )

  =>
  (assert (advice (result "call provider")))
  (printout t "call aa" crlf)
)

(defrule r3
  (answer (ident starter) (text no) )
  (answer (ident light) (text no) )

  =>
  (assert (advice (result "charge battery")))
  (printout t "charge battery" crlf)
)

(defrule r4
  (answer (ident starter) (text no) )
  (answer (ident light) (text yes) )
  (answer (ident solenoid) (text yes) )
  (answer (ident terminals) (text yes) )

  =>
  (assert (dvice (result "replace starter")))
  (printout t "replace starter" crlf)
)

(defrule r5
  (answer (ident starter) (text no) )
  (answer (ident light) (text yes) )
  (answer (ident solenoid) (text yes) )
  (answer (ident terminals) (text no) )

  =>
  (assert (advice (result "clean terminals")))
  (printout t "clean terminals" crlf)
)

```

```

(defrule r6
  (answer (ident starter) (text no) )
  (answer (ident light) (text yes) )
  (answer (ident solenoid) (text no) )
  (answer (ident fuse) (text no) )

  =>
  (assert (advice (result "replace fuse"))))
  (printout t "replace fuse" crlf)
)

(defrule r7
  (answer (ident starter) (text no) )
  (answer (ident light) (text yes) )
  (answer (ident solenoid) (text no) )
  (answer (ident fuse) (text yes) )
  =>
  (assert (advice (result "replace solenoid"))))
  (printout t "replace solenoid" crlf)
)

```

See the figure.

```

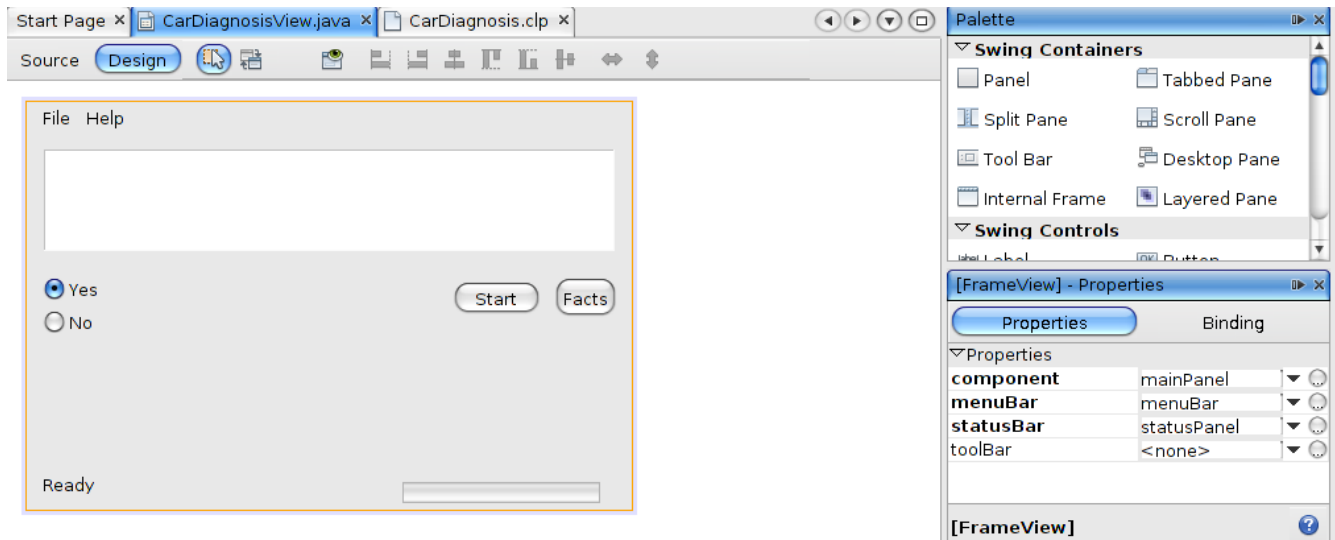
Start Page x CarDiagnosisView.java x CarDiagnosis.clp x
1  ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
2  ::::::;          Car Diagnosis Example
3  ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
4
5
6  (deftemplate question
7    "A question the application may ask"
8    (slot text)      ;; The question itself
9    (slot ident)     ;; The "name" of the question
10
11  (deftemplate answer
12    (slot ident)
13    (slot text))
14
15  (deftemplate advice
16    (slot result)
17  )
18  ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
19  ;; backward chaining answer (my goal)
20  (do-backward-chaining answer)
21  ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
22  (def facts question-data
23    (question (ident starter) (text "Does the car's starter turningg (yes or no)? "))
24    (question (ident petrol) (text "Do you have any petrol (yes or no)? "))
25    (question (ident light) (text "Are the lights working (yes or no)? "))

```

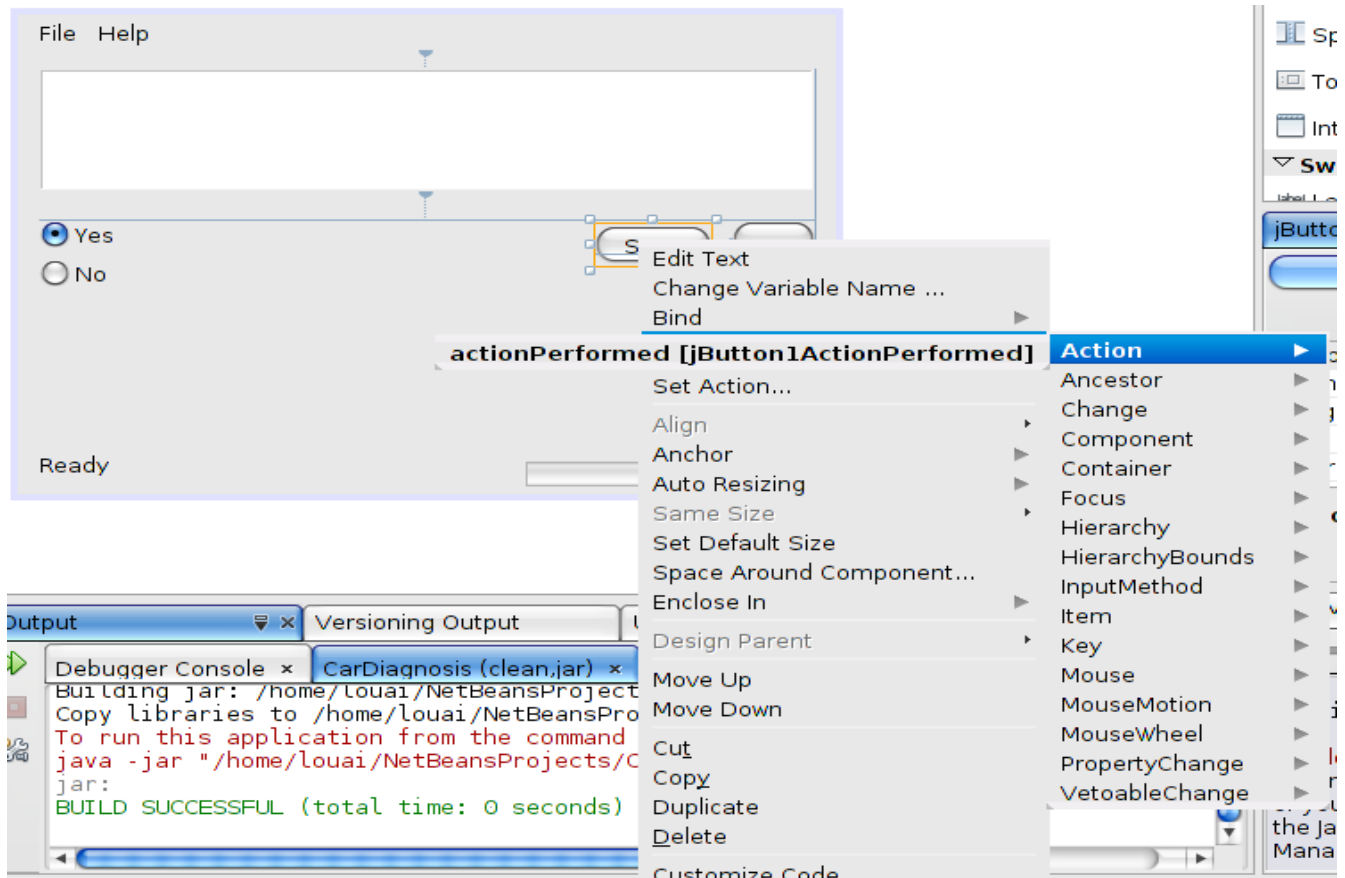
#### (4) Design your GUI Form

Hint:

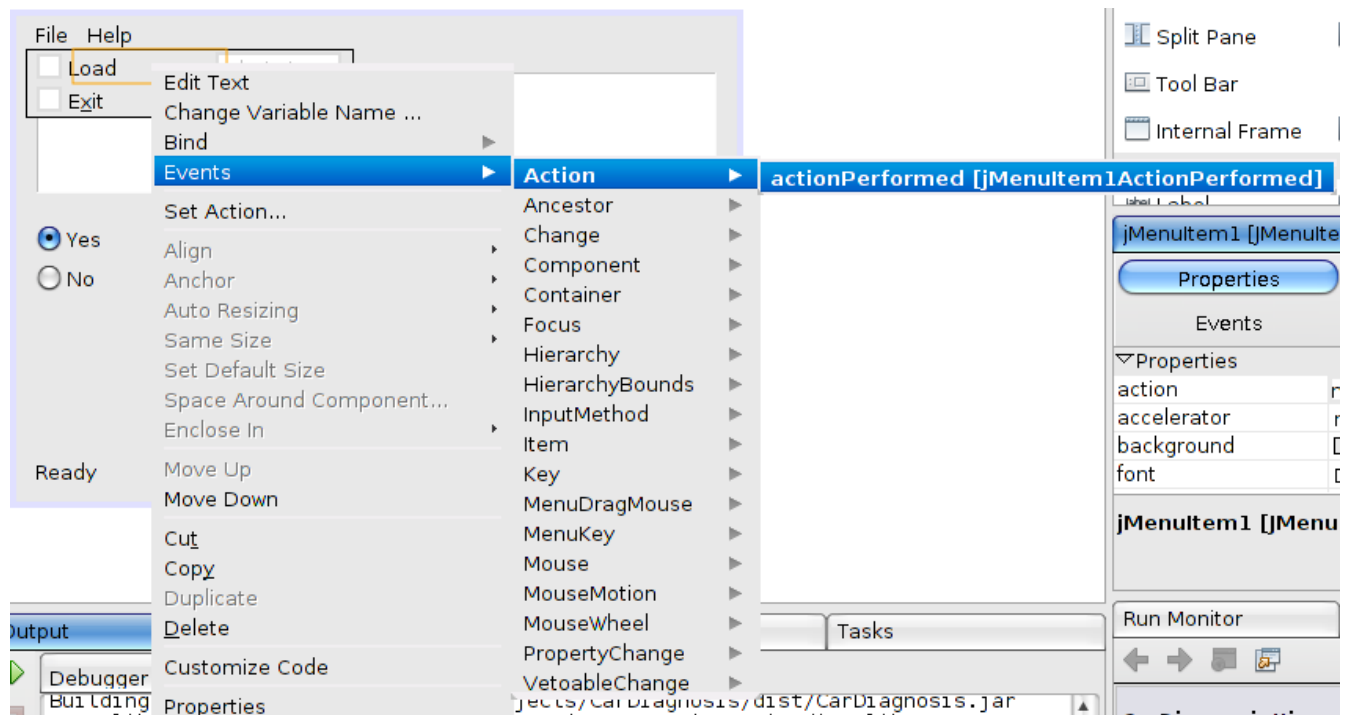
- Use palette window menu to drag and drop the tools.
- Properties window menu to show the properties of each object in GUI.



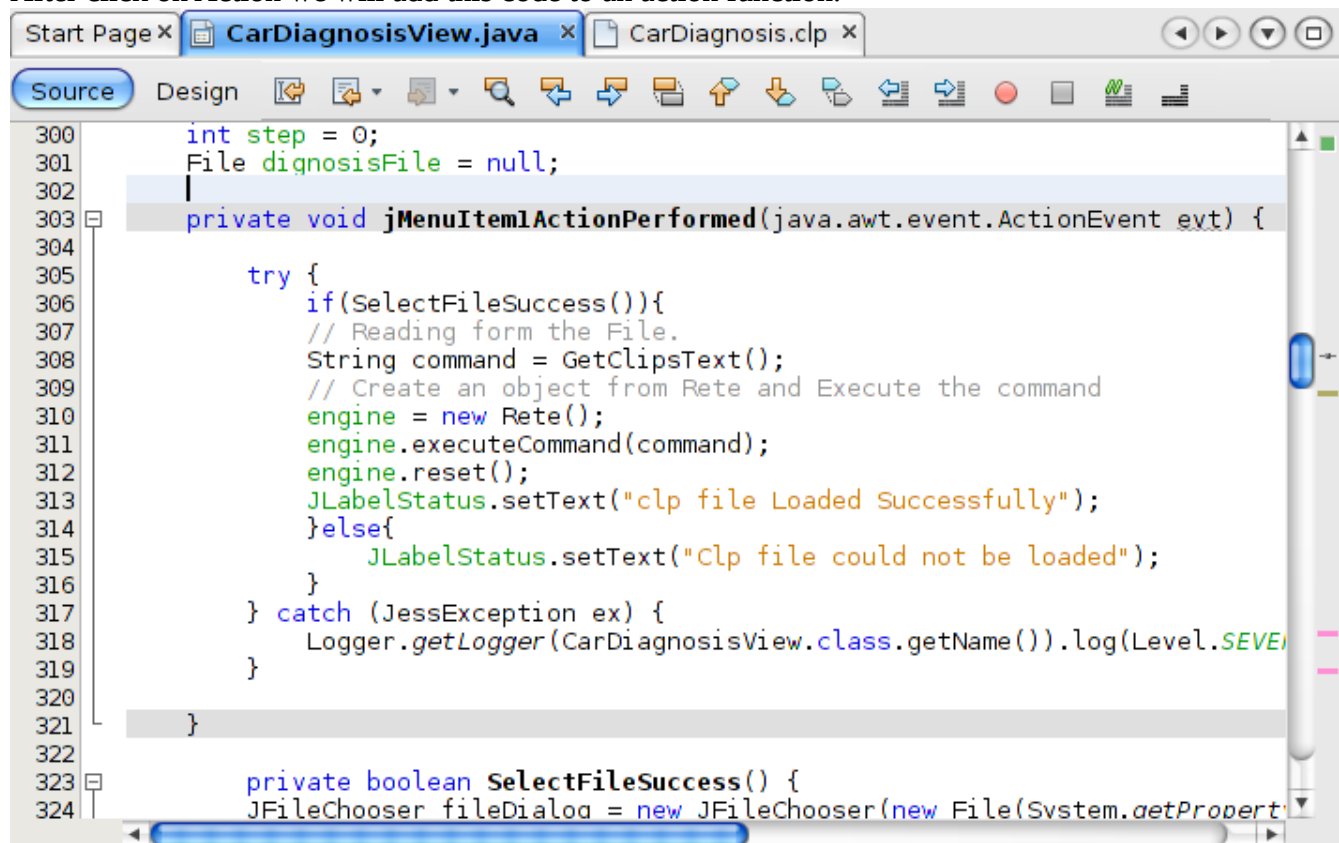
#### (5) Add event to GUI .



Now we want to add event to each object in Form as needed. To add event we click right-hand mouse to the object and then → Click on events → chose the event type we wish to fire, you will notice that I source code will be shown to write your code . For example we want to load clp file to our program.

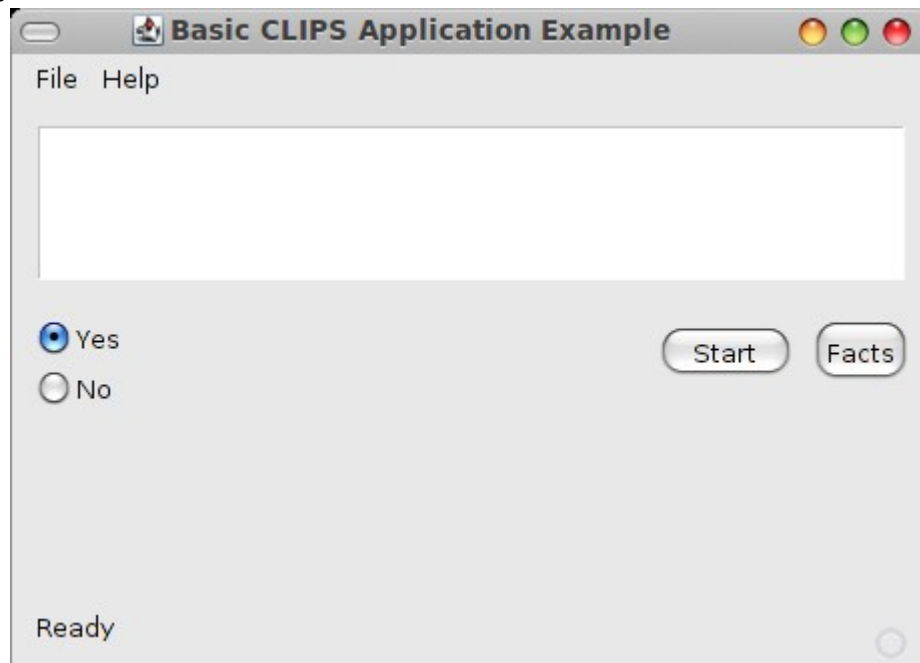


After click on Action we will add this code to an action function.

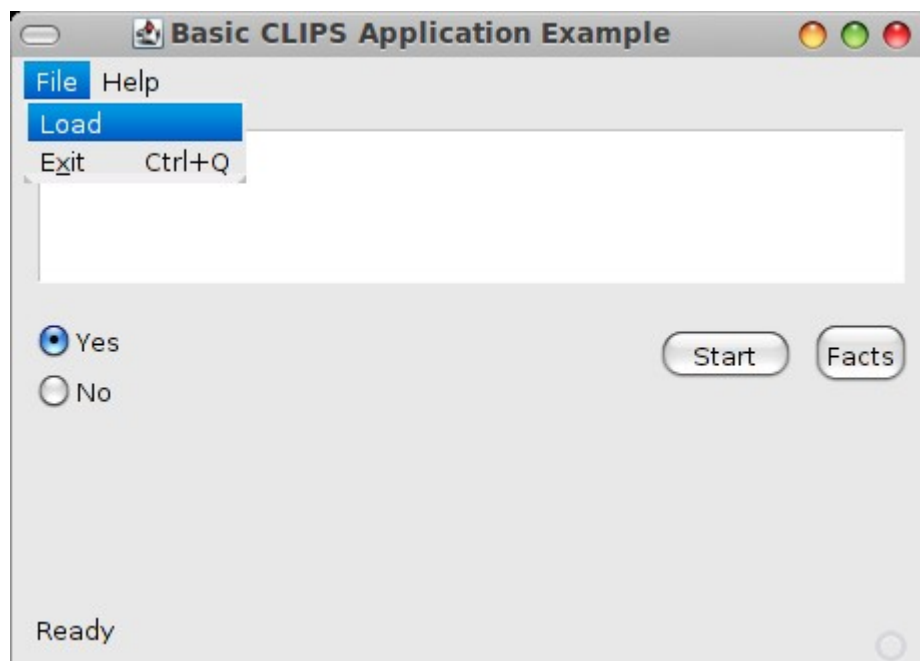


(6) Build and Run.

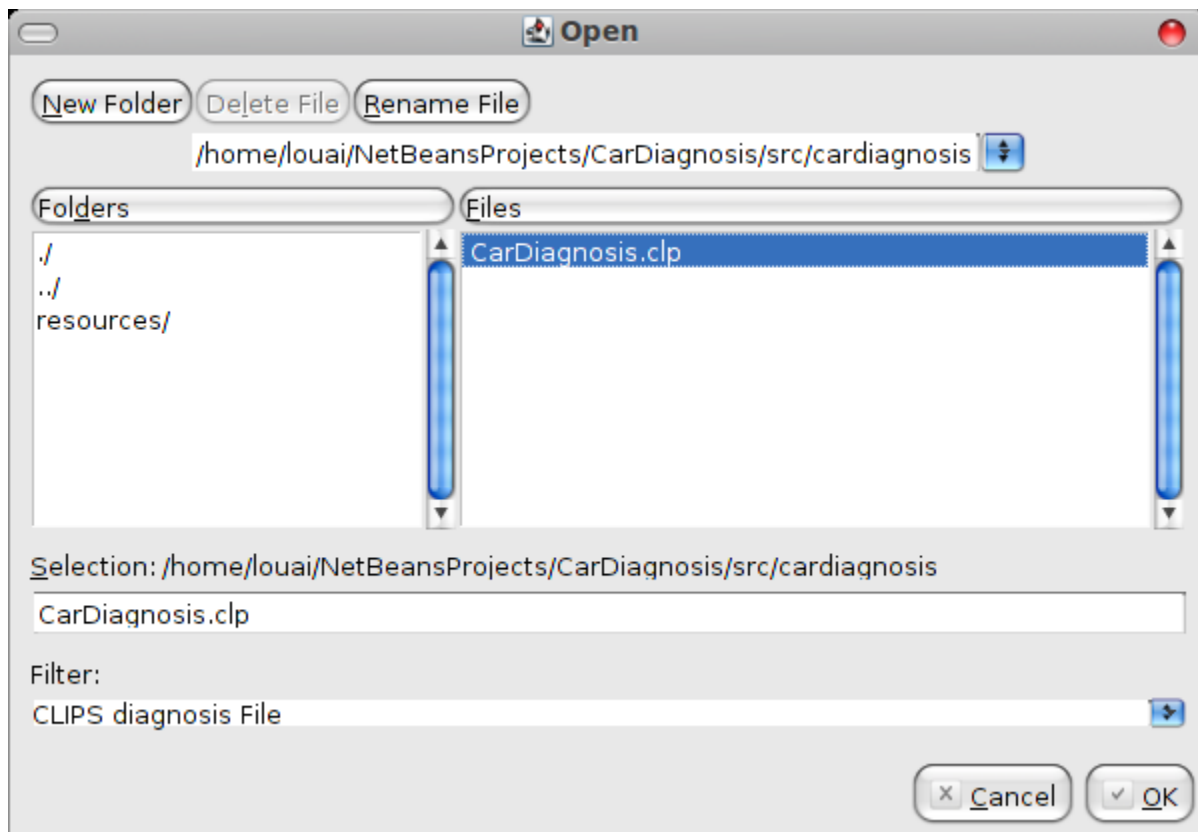
Run the project



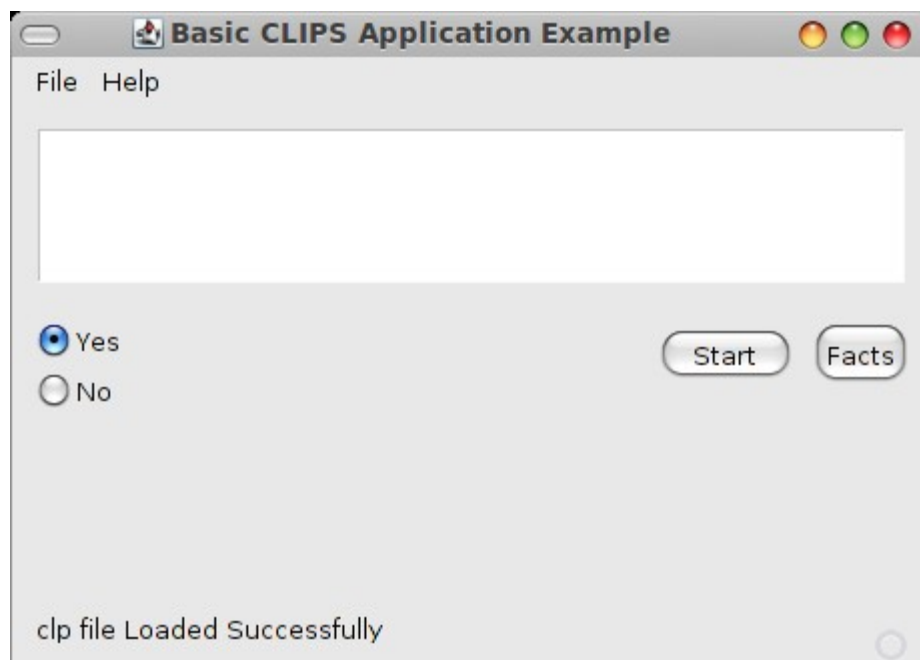
load the file .clp



Chose the clp file we create in the following directory src/cardagnosis

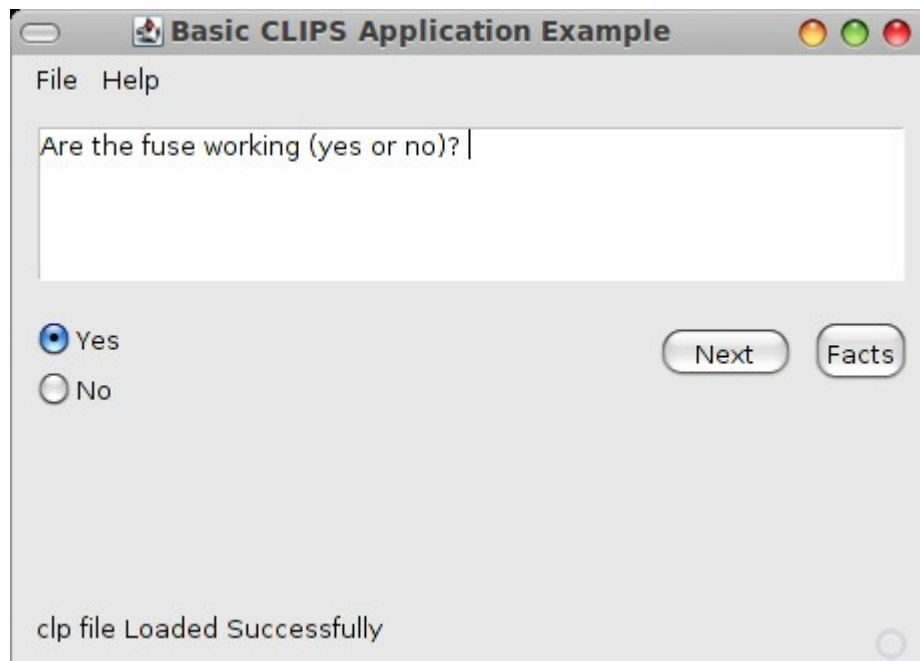


Notice that after loading the fie in status bar notification will be shown.



Click on Start button.

Answer a Question appear in text area.



An advice will be shown after answering all question.

